

Visualgo CS162

1.0.0

Generated by Doxygen 1.9.6



<b>1 Visualgo_CS162</b>	<b>1</b>
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List . . . . .	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy . . . . .	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List . . . . .	7
<b>5 File Index</b>	<b>9</b>
5.1 File List . . . . .	9
<b>6 Namespace Documentation</b>	<b>11</b>
6.1 constants Namespace Reference . . . . .	11
6.1.1 Function Documentation . . . . .	11
6.1.1.1 clickGray() . . . . .	12
6.1.1.2 clickGreen() . . . . .	12
6.1.1.3 hoverGray() . . . . .	12
6.1.1.4 hoverGreen() . . . . .	12
6.1.1.5 transparentGreen() . . . . .	12
6.1.2 Variable Documentation . . . . .	12
6.1.2.1 fontPath . . . . .	13
6.1.2.2 titleWindow . . . . .	13
6.2 constants::Arrow Namespace Reference . . . . .	13
6.2.1 Function Documentation . . . . .	13
6.2.1.1 defaultScaleRectangle() . . . . .	13
6.2.1.2 sizeRectangle() . . . . .	13
6.3 constants::ControlMenu Namespace Reference . . . . .	13
6.3.1 Enumeration Type Documentation . . . . .	14
6.3.1.1 Button . . . . .	14
6.3.2 Variable Documentation . . . . .	14
6.3.2.1 BUTTON_COUNT . . . . .	14
6.3.2.2 BUTTON_NAME_SIZE . . . . .	14
6.3.2.3 BUTTON_NAMES . . . . .	15
6.3.2.4 TEXT_SIZE . . . . .	15
6.4 constants::Highlighter Namespace Reference . . . . .	15
6.4.1 Function Documentation . . . . .	15
6.4.1.1 codeScale() . . . . .	15
6.5 constants::Highlighter::DLL Namespace Reference . . . . .	15
6.5.1 Variable Documentation . . . . .	16
6.5.1.1 CODES_PATH . . . . .	16
6.6 constants::Highlighter::SLL Namespace Reference . . . . .	16

6.6.1 Variable Documentation . . . . .	16
6.6.1.1 CODES_PATH . . . . .	16
6.7 constants::LinkedList Namespace Reference . . . . .	16
6.8 constants::MenuArray Namespace Reference . . . . .	16
6.8.1 Enumeration Type Documentation . . . . .	17
6.8.1.1 Button . . . . .	17
6.8.1.2 Type . . . . .	17
6.8.2 Variable Documentation . . . . .	18
6.8.2.1 BUTTON_COUNT . . . . .	18
6.8.2.2 BUTTON_NAME_SIZE . . . . .	18
6.8.2.3 BUTTON_NAMES . . . . .	18
6.9 constants::MenuArray::AddMode Namespace Reference . . . . .	18
6.9.1 Enumeration Type Documentation . . . . .	18
6.9.1.1 Textbox . . . . .	18
6.9.2 Variable Documentation . . . . .	19
6.9.2.1 TEXTBOX_COUNT . . . . .	19
6.9.2.2 TEXTBOX_LENGTH . . . . .	19
6.9.2.3 TEXTBOX_NAMES . . . . .	19
6.10 constants::MenuArray::AllocateMode Namespace Reference . . . . .	20
6.10.1 Enumeration Type Documentation . . . . .	20
6.10.1.1 Textbox . . . . .	20
6.10.2 Variable Documentation . . . . .	20
6.10.2.1 TEXTBOX_COUNT . . . . .	20
6.10.2.2 TEXTBOX_LENGTH . . . . .	20
6.10.2.3 TEXTBOX_NAME . . . . .	21
6.11 constants::MenuArray::CreateMode Namespace Reference . . . . .	21
6.11.1 Enumeration Type Documentation . . . . .	21
6.11.1.1 Button . . . . .	21
6.11.2 Variable Documentation . . . . .	21
6.11.2.1 BUTTON_COUNT . . . . .	22
6.11.2.2 BUTTON_NAMES . . . . .	22
6.11.2.3 NAME_SIZE . . . . .	22
6.11.2.4 TEXTBOX_COUNT . . . . .	22
6.11.2.5 TEXTBOX_LENGTH . . . . .	22
6.11.2.6 TEXTBOX_NAMES . . . . .	23
6.12 constants::MenuArray::DeleteMode Namespace Reference . . . . .	23
6.12.1 Enumeration Type Documentation . . . . .	23
6.12.1.1 Textbox . . . . .	23
6.12.2 Variable Documentation . . . . .	23
6.12.2.1 TEXTBOX_COUNT . . . . .	24
6.12.2.2 TEXTBOX_LENGTH . . . . .	24
6.12.2.3 TEXTBOX_NAME . . . . .	24

6.13 constants::MenuArray::SearchMode Namespace Reference	24
6.13.1 Enumeration Type Documentation	24
6.13.1.1 Textbox	24
6.13.2 Variable Documentation	25
6.13.2.1 TEXTBOX_COUNT	25
6.13.2.2 TEXTBOX_LENGTH	25
6.13.2.3 TEXTBOX_NAME	25
6.14 constants::MenuArray::UpdateMode Namespace Reference	25
6.14.1 Enumeration Type Documentation	25
6.14.1.1 Textbox	25
6.14.2 Variable Documentation	26
6.14.2.1 TEXTBOX_COUNT	26
6.14.2.2 TEXTBOX_LENGTH	26
6.14.2.3 TEXTBOX_NAMES	26
6.15 constants::MenuDataStructure Namespace Reference	27
6.15.1 Enumeration Type Documentation	27
6.15.1.1 Button	27
6.15.2 Variable Documentation	27
6.15.2.1 BUTTON_COUNT	28
6.15.2.2 BUTTON_NAME_SIZE	28
6.15.2.3 BUTTON_NAMES	28
6.16 constants::MenuDataStructure::CreateMode Namespace Reference	28
6.16.1 Enumeration Type Documentation	28
6.16.1.1 Button	28
6.16.2 Variable Documentation	29
6.16.2.1 BUTTON_COUNT	29
6.16.2.2 BUTTON_NAMES	29
6.16.2.3 NAME_SIZE	29
6.16.2.4 TEXTBOX_COUNT	30
6.16.2.5 TEXTBOX_LENGTH	30
6.16.2.6 TEXTBOX_NAMES	30
6.17 constants::MenuDataStructure::PushMode Namespace Reference	30
6.17.1 Enumeration Type Documentation	30
6.17.1.1 Textbox	30
6.17.2 Variable Documentation	31
6.17.2.1 TEXTBOX_COUNT	31
6.17.2.2 TEXTBOX_LENGTH	31
6.17.2.3 TEXTBOX_NAME	31
6.18 constants::MenuLinkedList Namespace Reference	31
6.18.1 Enumeration Type Documentation	32
6.18.1.1 Button	32
6.18.2 Variable Documentation	32

6.18.2.1	BUTTON_COUNT	32
6.18.2.2	BUTTON_NAME_SIZE	33
6.18.2.3	BUTTON_NAMES	33
6.19	constants::MenuLinkedList::AddMode Namespace Reference	33
6.19.1	Enumeration Type Documentation	33
6.19.1.1	Textbox	33
6.19.2	Variable Documentation	34
6.19.2.1	TEXTBOX_COUNT	34
6.19.2.2	TEXTBOX_LENGTH	34
6.19.2.3	TEXTBOX_NAMES	34
6.20	constants::MenuLinkedList::CreateMode Namespace Reference	34
6.20.1	Enumeration Type Documentation	35
6.20.1.1	Button	35
6.20.2	Variable Documentation	35
6.20.2.1	BUTTON_COUNT	35
6.20.2.2	BUTTON_NAMES	35
6.20.2.3	NAME_SIZE	36
6.20.2.4	TEXTBOX_COUNT	36
6.20.2.5	TEXTBOX_LENGTH	36
6.20.2.6	TEXTBOX_NAMES	36
6.21	constants::MenuLinkedList::DeleteMode Namespace Reference	36
6.21.1	Enumeration Type Documentation	36
6.21.1.1	Textbox	36
6.21.2	Variable Documentation	37
6.21.2.1	TEXTBOX_COUNT	37
6.21.2.2	TEXTBOX_LENGTH	37
6.21.2.3	TEXTBOX_NAME	37
6.22	constants::MenuLinkedList::SearchMode Namespace Reference	37
6.22.1	Enumeration Type Documentation	37
6.22.1.1	Textbox	37
6.22.2	Variable Documentation	38
6.22.2.1	TEXTBOX_COUNT	38
6.22.2.2	TEXTBOX_LENGTH	38
6.22.2.3	TEXTBOX_NAME	38
6.23	constants::MenuLinkedList::UpdateMode Namespace Reference	38
6.23.1	Enumeration Type Documentation	38
6.23.1.1	Textbox	38
6.23.2	Variable Documentation	39
6.23.2.1	TEXTBOX_COUNT	39
6.23.2.2	TEXTBOX_LENGTH	39
6.23.2.3	TEXTBOX_NAMES	39
6.24	constants::NodeInfo Namespace Reference	40

6.25 constants::sceneVariables Namespace Reference	40
6.25.1 Enumeration Type Documentation	40
6.25.1.1 Scene	40
6.25.2 Variable Documentation	40
6.25.2.1 NAME_MODE_BUTTON	41
6.25.2.2 SCENE_COUNT	41
6.25.2.3 SCENE_NAMES	41
6.26 constants::Square Namespace Reference	41
6.27 constants::TitleNode Namespace Reference	41
6.28 pfd Namespace Reference	41
6.28.1 Enumeration Type Documentation	42
6.28.1.1 button	42
6.28.1.2 choice	43
6.28.1.3 icon	43
6.28.1.4 opt	43
6.28.2 Function Documentation	44
6.28.2.1 operator&()	44
6.28.2.2 operator<<()	44
6.28.2.3 operator"   ()	44
6.29 pfd::internal Namespace Reference	45
6.30 Random Namespace Reference	45
6.31 sf Namespace Reference	45
<b>7 Class Documentation</b>	<b>47</b>
7.1 Array Class Reference	47
7.1.1 Detailed Description	48
7.1.2 Member Enumeration Documentation	48
7.1.2.1 TypeArray	48
7.1.3 Constructor & Destructor Documentation	48
7.1.3.1 Array() [1/3]	49
7.1.3.2 Array() [2/3]	49
7.1.3.3 Array() [3/3]	49
7.1.3.4 ~Array()	49
7.1.4 Member Function Documentation	49
7.1.4.1 addSquare()	50
7.1.4.2 allocateSquare()	50
7.1.4.3 createArray() [1/2]	51
7.1.4.4 createArray() [2/2]	51
7.1.4.5 deleteSquare()	52
7.1.4.6 findValue()	52
7.1.4.7 getSize()	52
7.1.4.8 getSquaresSize()	52

---

7.1.4.9	init()	53
7.1.4.10	initHighlighter()	53
7.1.4.11	processControlMenu()	53
7.1.4.12	render()	54
7.1.4.13	renderHighlighter()	54
7.1.4.14	resetEvents()	54
7.1.4.15	searchSquare()	54
7.1.4.16	setSpeed()	55
7.1.4.17	toggleLines()	55
7.1.4.18	update()	55
7.1.4.19	updateAnimation()	55
7.1.4.20	updateSquare()	56
7.2	Arrow Class Reference	56
7.2.1	Detailed Description	57
7.2.2	Constructor & Destructor Documentation	57
7.2.2.1	Arrow()	57
7.2.3	Member Function Documentation	58
7.2.3.1	autoRotate()	58
7.2.3.2	autoScale()	58
7.2.3.3	hide()	59
7.2.3.4	render()	59
7.2.3.5	resetColor()	59
7.2.3.6	setMid()	59
7.2.3.7	setPositions()	60
7.2.3.8	setStart()	60
7.2.3.9	show()	60
7.2.3.10	toggleActiveColor()	60
7.2.4	Member Data Documentation	61
7.2.4.1	arrowSprite	61
7.2.4.2	arrowTexture	61
7.2.4.3	hasSetMid	61
7.2.4.4	length	61
7.2.4.5	points	61
7.3	BackArrow Class Reference	62
7.3.1	Detailed Description	62
7.3.2	Constructor & Destructor Documentation	62
7.3.2.1	BackArrow()	63
7.3.3	Member Function Documentation	63
7.3.3.1	autoRotate()	63
7.3.3.2	autoScale()	64
7.3.3.3	hide()	64
7.3.3.4	render()	64



7.3.3.5 resetColor()	65
7.3.3.6 setPosition()	65
7.3.3.7 show()	65
7.3.3.8 toggleActiveColorNode()	65
7.4 BaseDraw Class Reference	66
7.4.1 Detailed Description	66
7.4.2 Constructor & Destructor Documentation	66
7.4.2.1 BaseDraw()	66
7.4.3 Member Function Documentation	66
7.4.3.1 render()	66
7.4.4 Member Data Documentation	67
7.4.4.1 window	67
7.5 BaseScene Class Reference	67
7.5.1 Detailed Description	68
7.5.2 Constructor & Destructor Documentation	68
7.5.2.1 BaseScene()	68
7.5.3 Member Function Documentation	68
7.5.3.1 createModeButton()	69
7.5.3.2 pollEvent()	69
7.5.3.3 render()	69
7.5.3.4 setWindow()	69
7.5.3.5 update()	70
7.5.4 Member Data Documentation	70
7.5.4.1 controlMenu	70
7.5.4.2 isDemoCodeOpen	70
7.5.4.3 isMenuOpen	70
7.5.4.4 modeButton	70
7.5.4.5 window	71
7.6 Button Class Reference	71
7.6.1 Detailed Description	71
7.6.2 Constructor & Destructor Documentation	71
7.6.2.1 Button() [1/2]	71
7.6.2.2 Button() [2/2]	72
7.6.3 Member Function Documentation	72
7.6.3.1 checkClicked()	72
7.6.3.2 getPosition()	72
7.6.3.3 getSize()	72
7.6.3.4 getTextString()	72
7.6.3.5 pollEvent()	72
7.6.3.6 render()	73
7.6.3.7 setColor()	73
7.6.3.8 update()	73

7.7 CLLScene Class Reference . . . . .	73
7.7.1 Detailed Description . . . . .	74
7.7.2 Constructor & Destructor Documentation . . . . .	74
7.7.2.1 CLLScene() . . . . .	74
7.7.3 Member Function Documentation . . . . .	74
7.7.3.1 addModeEvents() . . . . .	75
7.7.3.2 deleteModeEvents() . . . . .	76
7.7.3.3 pollEvent() . . . . .	78
7.7.3.4 render() . . . . .	79
7.7.3.5 reset() . . . . .	79
7.7.3.6 searchModeEvents() . . . . .	79
7.7.3.7 update() . . . . .	80
7.7.3.8 updateModeEvents() . . . . .	81
7.8 ControlMenu Class Reference . . . . .	82
7.8.1 Detailed Description . . . . .	83
7.8.2 Member Enumeration Documentation . . . . .	83
7.8.2.1 StatusCode . . . . .	83
7.8.3 Constructor & Destructor Documentation . . . . .	84
7.8.3.1 ControlMenu() . . . . .	84
7.8.3.2 ~ControlMenu() . . . . .	84
7.8.4 Member Function Documentation . . . . .	84
7.8.4.1 getSpeed() . . . . .	84
7.8.4.2 getStatus() . . . . .	85
7.8.4.3 pollEvents() . . . . .	85
7.8.4.4 render() . . . . .	85
7.8.4.5 reset() . . . . .	86
7.8.4.6 update() . . . . .	86
7.8.5 Member Data Documentation . . . . .	86
7.8.5.1 status . . . . .	86
7.9 CustomTextbox Class Reference . . . . .	86
7.9.1 Detailed Description . . . . .	87
7.9.2 Constructor & Destructor Documentation . . . . .	87
7.9.2.1 CustomTextbox() . . . . .	87
7.9.2.2 ~CustomTextbox() . . . . .	87
7.9.3 Member Function Documentation . . . . .	88
7.9.3.1 getTextString() . . . . .	88
7.9.3.2 pollEvent() . . . . .	88
7.9.3.3 render() . . . . .	88
7.9.3.4 resetInput() . . . . .	88
7.9.3.5 update() . . . . .	89
7.10 pfd::internal::dialog Class Reference . . . . .	89
7.10.1 Detailed Description . . . . .	90

7.10.2 Constructor & Destructor Documentation	90
7.10.2.1 dialog()	90
7.10.3 Member Function Documentation	90
7.10.3.1 buttons_to_name()	91
7.10.3.2 desktop_helper()	91
7.10.3.3 get_icon_name()	91
7.10.3.4 kill()	92
7.10.3.5 osascript_quote()	92
7.10.3.6 powershell_quote()	92
7.10.3.7 ready()	92
7.10.3.8 shell_quote()	92
7.10.4 Member Data Documentation	93
7.10.4.1 m_async	93
7.11 DLLScene Class Reference	93
7.11.1 Detailed Description	94
7.11.2 Constructor & Destructor Documentation	94
7.11.2.1 DLLScene()	94
7.11.3 Member Function Documentation	94
7.11.3.1 addModeEvents()	95
7.11.3.2 deleteModeEvents()	98
7.11.3.3 pollEvent()	100
7.11.3.4 render()	101
7.11.3.5 reset()	101
7.11.3.6 searchModeEvents()	101
7.11.3.7 update()	102
7.11.3.8 updateModeEvents()	104
7.12 DynamicArrayScene Class Reference	105
7.12.1 Detailed Description	106
7.12.2 Constructor & Destructor Documentation	106
7.12.2.1 DynamicArrayScene()	106
7.12.3 Member Function Documentation	106
7.12.3.1 addModeEvents()	107
7.12.3.2 allocateModeEvents()	108
7.12.3.3 deleteModeEvents()	109
7.12.3.4 pollEvent()	110
7.12.3.5 render()	110
7.12.3.6 reset()	111
7.12.3.7 searchModeEvents()	111
7.12.3.8 update()	112
7.12.3.9 updateModeEvents()	113
7.13 EventAnimation Class Reference	114
7.13.1 Detailed Description	114

7.13.2 Constructor & Destructor Documentation	114
7.13.2.1 EventAnimation()	115
7.13.2.2 ~EventAnimation()	115
7.13.3 Member Function Documentation	115
7.13.3.1 operator=()	115
7.13.3.2 reset()	115
7.13.4 Member Data Documentation	116
7.13.4.1 colorArrows	116
7.13.4.2 colorNodes	116
7.13.4.3 eventSquares	116
7.13.4.4 eventSquaresTemp	116
7.13.4.5 hiddenArrows	116
7.13.4.6 indexBackArrow	117
7.13.4.7 isPrintNormal	117
7.13.4.8 isPrintPreVal	117
7.13.4.9 isShowBackArrow	117
7.13.4.10 lines	117
7.13.4.11 statusChosenNode	117
7.13.4.12 titleNodes	118
7.14 EventSquare Struct Reference	118
7.14.1 Detailed Description	118
7.14.2 Constructor & Destructor Documentation	118
7.14.2.1 EventSquare()	118
7.14.2.2 ~EventSquare()	118
7.14.3 Member Data Documentation	119
7.14.3.1 isPrintPreVal	119
7.14.3.2 status	119
7.14.3.3 title	119
7.15 pfd::internal::executor Class Reference	119
7.15.1 Detailed Description	120
7.15.2 Constructor & Destructor Documentation	120
7.15.2.1 ~executor()	120
7.15.3 Member Function Documentation	120
7.15.3.1 kill()	120
7.15.3.2 ready()	121
7.15.3.3 result()	121
7.15.3.4 start_process()	122
7.15.3.5 stop()	122
7.15.4 Friends And Related Function Documentation	122
7.15.4.1 dialog	123
7.16 pfd::internal::file_dialog Class Reference	123
7.16.1 Detailed Description	124

7.16.2 Member Enumeration Documentation . . . . .	124
7.16.2.1 type . . . . .	124
7.16.3 Constructor & Destructor Documentation . . . . .	125
7.16.3.1 file_dialog() . . . . .	125
7.16.4 Member Function Documentation . . . . .	129
7.16.4.1 string_result() . . . . .	129
7.16.4.2 vector_result() . . . . .	129
7.17 Highlighter Class Reference . . . . .	129
7.17.1 Detailed Description . . . . .	130
7.17.2 Constructor & Destructor Documentation . . . . .	130
7.17.2.1 Highlighter() . . . . .	130
7.17.3 Member Function Documentation . . . . .	130
7.17.3.1 render() . . . . .	130
7.17.3.2 resetToggle() . . . . .	131
7.17.3.3 toggle() . . . . .	131
7.18 LinkedList Class Reference . . . . .	131
7.18.1 Detailed Description . . . . .	132
7.18.2 Member Enumeration Documentation . . . . .	132
7.18.2.1 TypeLinkedList . . . . .	132
7.18.3 Constructor & Destructor Documentation . . . . .	132
7.18.3.1 LinkedList() [1/3] . . . . .	132
7.18.3.2 LinkedList() [2/3] . . . . .	133
7.18.3.3 LinkedList() [3/3] . . . . .	133
7.18.4 Member Function Documentation . . . . .	133
7.18.4.1 addNode() . . . . .	133
7.18.4.2 calculateEffectivePositions() . . . . .	134
7.18.4.3 clear() . . . . .	135
7.18.4.4 createLinkedList() [1/2] . . . . .	135
7.18.4.5 createLinkedList() [2/2] . . . . .	135
7.18.4.6 deleteNode() . . . . .	136
7.18.4.7 findValue() . . . . .	136
7.18.4.8 getPosNode() . . . . .	137
7.18.4.9 getSize() . . . . .	137
7.18.4.10 initHighlighter() . . . . .	137
7.18.4.11 processControlMenu() . . . . .	137
7.18.4.12 render() . . . . .	138
7.18.4.13 renderHighlighter() . . . . .	138
7.18.4.14 resetEvents() . . . . .	138
7.18.4.15 searchNode() . . . . .	139
7.18.4.16 setSpeed() . . . . .	139
7.18.4.17 toggleLines() . . . . .	139
7.18.4.18 update() . . . . .	139

7.18.4.19 updateAnimation()	140
7.18.4.20 updateNode()	141
7.19 MainMenu Class Reference	141
7.19.1 Detailed Description	142
7.19.2 Constructor & Destructor Documentation	142
7.19.2.1 MainMenu()	142
7.19.3 Member Function Documentation	142
7.19.3.1 pollEvent()	142
7.19.3.2 render()	143
7.19.3.3 update()	143
7.20 MenuArray Class Reference	143
7.20.1 Detailed Description	144
7.20.2 Constructor & Destructor Documentation	144
7.20.2.1 MenuArray()	144
7.20.2.2 ~MenuArray()	144
7.20.3 Member Function Documentation	144
7.20.3.1 getActiveCreateMode()	144
7.20.3.2 getActiveOptionsMenu()	144
7.20.3.3 getButton()	145
7.20.3.4 pollEvents()	145
7.20.3.5 render()	145
7.20.3.6 resetActiveOptionsMenu()	146
7.20.3.7 update()	146
7.20.4 Member Data Documentation	147
7.20.4.1 addModeValue	147
7.20.4.2 allocateModeValue	147
7.20.4.3 createModeValue	147
7.20.4.4 deleteModeValue	147
7.20.4.5 searchModeValue	147
7.20.4.6 updateModeValue	148
7.21 MenuDataStructure Class Reference	148
7.21.1 Detailed Description	148
7.21.2 Constructor & Destructor Documentation	148
7.21.2.1 MenuDataStructure()	149
7.21.2.2 ~MenuDataStructure()	149
7.21.3 Member Function Documentation	149
7.21.3.1 getActiveCreateMode()	149
7.21.3.2 getActiveOptionsMenu()	149
7.21.3.3 getButton()	149
7.21.3.4 pollEvents()	150
7.21.3.5 render()	150
7.21.3.6 resetActiveOptionsMenu()	150

7.21.3.7 resetActiveOptionsMenuOnly()	150
7.21.3.8 update()	151
7.21.4 Member Data Documentation	151
7.21.4.1 createModeValue	151
7.21.4.2 pushModeValue	151
7.22 MenuLinkedList Class Reference	151
7.22.1 Detailed Description	153
7.22.2 Constructor & Destructor Documentation	153
7.22.2.1 MenuLinkedList()	153
7.22.2.2 ~MenuLinkedList()	153
7.22.3 Member Function Documentation	153
7.22.3.1 getActiveCreateMode()	154
7.22.3.2 getActiveOptionsMenu()	154
7.22.3.3 getButton()	154
7.22.3.4 init()	154
7.22.3.5 initAddMode()	155
7.22.3.6 initButtons()	155
7.22.3.7 initCreateMode()	155
7.22.3.8 initDeleteMode()	156
7.22.3.9 initSearchMode()	156
7.22.3.10 initUpdateMode()	157
7.22.3.11 pollEventAddMode()	157
7.22.3.12 pollEventCreateMode()	157
7.22.3.13 pollEventDeleteMode()	158
7.22.3.14 pollEvents()	158
7.22.3.15 pollEventSearchMode()	158
7.22.3.16 pollEventUpdateMode()	159
7.22.3.17 render()	159
7.22.3.18 renderAddMode()	159
7.22.3.19 renderCreateMode()	160
7.22.3.20 renderDeleteMode()	160
7.22.3.21 renderSearchMode()	160
7.22.3.22 renderUpdateMode()	160
7.22.3.23 resetActiveOptionsMenu()	160
7.22.3.24 update()	161
7.22.3.25 updateAddMode()	161
7.22.3.26 updateCreateMode()	161
7.22.3.27 updateDeleteMode()	162
7.22.3.28 updateSearchMode()	162
7.22.3.29 updateUpdateMode()	163
7.22.4 Member Data Documentation	163
7.22.4.1 activeAddMode	163

7.22.4.2 activeCreateMode	163
7.22.4.3 activeOptionsMenu	164
7.22.4.4 activeUpdateMode	164
7.22.4.5 addModeValue	164
7.22.4.6 addTextbox	164
7.22.4.7 buttons	164
7.22.4.8 createModeValue	164
7.22.4.9 createTextbox	165
7.22.4.10 deleteModeValue	165
7.22.4.11 deleteTextbox	165
7.22.4.12 isOpenFileDialog	165
7.22.4.13 searchModeValue	165
7.22.4.14 searchTextbox	165
7.22.4.15 subCreateMode	166
7.22.4.16 updateModeValue	166
7.22.4.17 updateTextbox	166
7.22.4.18 window	166
7.23 pfd::message Class Reference	166
7.23.1 Detailed Description	168
7.23.2 Constructor & Destructor Documentation	168
7.23.2.1 message()	168
7.23.3 Member Function Documentation	170
7.23.3.1 result()	170
7.24 MousePosition Class Reference	171
7.24.1 Detailed Description	171
7.24.2 Member Function Documentation	172
7.24.2.1 updateMousePosition()	172
7.24.3 Member Data Documentation	172
7.24.3.1 mousePos	172
7.24.3.2 mousePosView	172
7.24.3.3 relativeWindow	172
7.25 NodeInfo Class Reference	173
7.25.1 Detailed Description	174
7.25.2 Member Enumeration Documentation	174
7.25.2.1 ArrowType	174
7.25.2.2 StatusNode	174
7.25.2.3 TypeNode	175
7.25.3 Constructor & Destructor Documentation	175
7.25.3.1 NodeInfo()	175
7.25.3.2 ~NodeInfo()	176
7.25.4 Member Function Documentation	176
7.25.4.1 destroyArrow()	176



7.25.4.2	<a href="#">getPosition()</a>	176
7.25.4.3	<a href="#">getStatusNode()</a>	176
7.25.4.4	<a href="#">getValue()</a>	177
7.25.4.5	<a href="#">hide()</a>	177
7.25.4.6	<a href="#">initArrow()</a>	177
7.25.4.7	<a href="#">reInitPos()</a>	177
7.25.4.8	<a href="#">reInitPreVal()</a>	178
7.25.4.9	<a href="#">render()</a>	178
7.25.4.10	<a href="#">reset()</a>	178
7.25.4.11	<a href="#">resetColorArrow()</a>	178
7.25.4.12	<a href="#">resetColorNode()</a>	179
7.25.4.13	<a href="#">resetTitle()</a>	179
7.25.4.14	<a href="#">setArrows()</a>	179
7.25.4.15	<a href="#">setEffectivePosition()</a>	179
7.25.4.16	<a href="#">setNodeInChain()</a>	179
7.25.4.17	<a href="#">setNodeOutside()</a>	180
7.25.4.18	<a href="#">setNodeVisible()</a>	180
7.25.4.19	<a href="#">setPrintNormal()</a>	180
7.25.4.20	<a href="#">setPrintPreVal()</a>	180
7.25.4.21	<a href="#">setTitle()</a>	180
7.25.4.22	<a href="#">setValue()</a>	181
7.25.4.23	<a href="#">show()</a>	181
7.25.4.24	<a href="#">toggleActiveColorArrow()</a>	181
7.25.4.25	<a href="#">toggleActiveColorNode()</a>	181
7.25.4.26	<a href="#">updateArrows()</a>	182
7.25.4.27	<a href="#">updateNode()</a>	182
7.26	<a href="#">pfd::notify Class Reference</a>	182
7.26.1	<a href="#">Detailed Description</a>	184
7.26.2	<a href="#">Constructor &amp; Destructor Documentation</a>	184
7.26.2.1	<a href="#">notify()</a>	184
7.27	<a href="#">pfd::open_file Class Reference</a>	185
7.27.1	<a href="#">Detailed Description</a>	187
7.27.2	<a href="#">Constructor &amp; Destructor Documentation</a>	187
7.27.2.1	<a href="#">open_file()</a> [1/2]	187
7.27.2.2	<a href="#">open_file()</a> [2/2]	187
7.27.3	<a href="#">Member Function Documentation</a>	188
7.27.3.1	<a href="#">result()</a>	188
7.28	<a href="#">pfd::path Class Reference</a>	188
7.28.1	<a href="#">Detailed Description</a>	188
7.28.2	<a href="#">Member Function Documentation</a>	188
7.28.2.1	<a href="#">home()</a>	189
7.28.2.2	<a href="#">separator()</a>	189

7.29 pfd::internal::platform Class Reference	190
7.29.1 Detailed Description	190
7.30 QueueScene Class Reference	190
7.30.1 Detailed Description	191
7.30.2 Constructor & Destructor Documentation	191
7.30.2.1 QueueScene()	191
7.30.3 Member Function Documentation	191
7.30.3.1 pollEvent()	192
7.30.3.2 popModeEvents()	192
7.30.3.3 pushModeEvents()	194
7.30.3.4 render()	195
7.30.3.5 reset()	196
7.30.3.6 update()	196
7.31 sf::RoundedRectangleShape Class Reference	197
7.31.1 Detailed Description	198
7.31.2 Constructor & Destructor Documentation	198
7.31.2.1 RoundedRectangleShape()	198
7.31.3 Member Function Documentation	199
7.31.3.1 getCornersRadius()	199
7.31.3.2 getPoint()	199
7.31.3.3 getPointCount()	200
7.31.3.4 getSize()	200
7.31.3.5 setCornerPointCount()	200
7.31.3.6 setCornersRadius()	201
7.31.3.7 setSize()	201
7.32 pfd::save_file Class Reference	202
7.32.1 Detailed Description	203
7.32.2 Constructor & Destructor Documentation	204
7.32.2.1 save_file() [1/2]	204
7.32.2.2 save_file() [2/2]	204
7.32.3 Member Function Documentation	204
7.32.3.1 result()	204
7.33 pfd::select_folder Class Reference	205
7.33.1 Detailed Description	206
7.33.2 Constructor & Destructor Documentation	206
7.33.2.1 select_folder()	206
7.33.3 Member Function Documentation	207
7.33.3.1 result()	207
7.34 pfd::settings Class Reference	207
7.34.1 Detailed Description	208
7.34.2 Member Enumeration Documentation	208
7.34.2.1 flag	208

7.34.3 Constructor & Destructor Documentation	208
7.34.3.1 settings()	209
7.34.4 Member Function Documentation	209
7.34.4.1 available()	209
7.34.4.2 check_program()	210
7.34.4.3 flags() [1/2]	210
7.34.4.4 flags() [2/2]	210
7.34.4.5 is_kdialog()	210
7.34.4.6 is_osascript()	211
7.34.4.7 is_zenity()	211
7.34.4.8 rescan()	211
7.34.4.9 verbose()	211
7.35 SingleNode Class Reference	212
7.35.1 Detailed Description	212
7.35.2 Constructor & Destructor Documentation	212
7.35.2.1 SingleNode()	213
7.35.3 Member Function Documentation	213
7.35.3.1 getPosition()	213
7.35.3.2 render()	213
7.35.3.3 resetColor()	214
7.35.3.4 setPosition()	214
7.35.3.5 setText()	214
7.35.3.6 toggleActiveColor()	214
7.35.4 Member Data Documentation	214
7.35.4.1 font	215
7.36 SLLScene Class Reference	215
7.36.1 Detailed Description	216
7.36.2 Constructor & Destructor Documentation	216
7.36.2.1 SLLScene()	216
7.36.3 Member Function Documentation	216
7.36.3.1 addModeEvents()	216
7.36.3.2 deleteModeEvents()	218
7.36.3.3 pollEvent()	220
7.36.3.4 render()	220
7.36.3.5 reset()	220
7.36.3.6 searchModeEvents()	221
7.36.3.7 update()	221
7.36.3.8 updateModeEvents()	223
7.37 Square Class Reference	224
7.37.1 Detailed Description	225
7.37.2 Member Enumeration Documentation	225
7.37.2.1 Status	225

7.37.3 Constructor & Destructor Documentation	225
7.37.3.1 Square()	225
7.37.4 Member Function Documentation	226
7.37.4.1 getPosition()	226
7.37.4.2 getStatus()	226
7.37.4.3 render()	226
7.37.4.4 resetColor()	226
7.37.4.5 setPosition()	227
7.37.4.6 setStatus()	227
7.37.4.7 setText()	227
7.37.5 Member Data Documentation	227
7.37.5.1 font	227
7.38 SquareInfo Class Reference	228
7.38.1 Detailed Description	228
7.38.2 Constructor & Destructor Documentation	228
7.38.2.1 SquareInfo()	229
7.38.2.2 ~SquareInfo()	229
7.38.3 Member Function Documentation	229
7.38.3.1 getStatus()	229
7.38.3.2 getValue()	229
7.38.3.3 render()	230
7.38.3.4 reset()	230
7.38.3.5 resetTitle()	230
7.38.3.6 setPrintPreVal()	230
7.38.3.7 setStatus()	231
7.38.3.8 setTitle()	231
7.38.3.9 setValue()	231
7.38.3.10 update()	231
7.39 StackScene Class Reference	232
7.39.1 Detailed Description	233
7.39.2 Constructor & Destructor Documentation	233
7.39.2.1 StackScene()	233
7.39.3 Member Function Documentation	233
7.39.3.1 pollEvent()	233
7.39.3.2 popModeEvents()	234
7.39.3.3 pushModeEvents()	235
7.39.3.4 render()	237
7.39.3.5 reset()	237
7.39.3.6 update()	238
7.40 StaticArrayScene Class Reference	239
7.40.1 Detailed Description	240
7.40.2 Constructor & Destructor Documentation	240

7.40.2.1 StaticArrayScene()	240
7.40.3 Member Function Documentation	240
7.40.3.1 addModeEvents()	241
7.40.3.2 deleteModeEvents()	241
7.40.3.3 pollEvent()	242
7.40.3.4 render()	243
7.40.3.5 reset()	243
7.40.3.6 searchModeEvents()	243
7.40.3.7 update()	244
7.40.3.8 updateModeEvents()	245
7.41 TextBox Class Reference	246
7.41.1 Detailed Description	246
7.41.2 Constructor & Destructor Documentation	246
7.41.2.1 TextBox()	246
7.41.3 Member Function Documentation	247
7.41.3.1 getBox()	247
7.41.3.2 getTextString()	247
7.41.3.3 pollEvent()	247
7.41.3.4 render()	248
7.41.3.5 resetInput()	248
7.41.3.6 update()	248
7.42 Vector< T > Class Template Reference	248
7.42.1 Detailed Description	249
7.42.2 Constructor & Destructor Documentation	249
7.42.2.1 Vector() [1/3]	249
7.42.2.2 Vector() [2/3]	250
7.42.2.3 Vector() [3/3]	250
7.42.2.4 ~Vector()	250
7.42.3 Member Function Documentation	250
7.42.3.1 assign()	250
7.42.3.2 at()	251
7.42.3.3 back()	251
7.42.3.4 begin()	251
7.42.3.5 clear()	251
7.42.3.6 data()	251
7.42.3.7 empty()	252
7.42.3.8 end()	252
7.42.3.9 erase() [1/2]	252
7.42.3.10 erase() [2/2]	252
7.42.3.11 front()	253
7.42.3.12 getCapacity()	253
7.42.3.13 insert()	253

7.42.3.14 operator=()	253
7.42.3.15 operator[]()	254
7.42.3.16 pop_back()	254
7.42.3.17 push_back()	254
7.42.3.18 resize()	254
7.42.3.19 size()	255
7.43 Window Class Reference	255
7.43.1 Detailed Description	256
7.43.2 Constructor & Destructor Documentation	256
7.43.2.1 Window()	256
7.43.2.2 ~Window()	256
7.43.3 Member Function Documentation	256
7.43.3.1 pollEvent()	256
7.43.3.2 render()	257
7.43.3.3 running()	257
7.43.3.4 update()	258
<b>8 File Documentation</b>	<b>259</b>
8.1 include/Constants.hpp File Reference	259
8.2 Constants.hpp	262
8.3 include/core/Array.cpp File Reference	267
8.4 Array.cpp	267
8.5 include/core/Array.hpp File Reference	271
8.6 Array.hpp	272
8.7 include/core/EventAnimation.cpp File Reference	272
8.8 EventAnimation.cpp	273
8.9 include/core/EventAnimation.hpp File Reference	273
8.10 EventAnimation.hpp	273
8.11 include/core/FileDialog.h File Reference	274
8.11.1 Macro Definition Documentation	275
8.11.1.1 _POSIX_C_SOURCE	275
8.11.1.2 PFD_HAS_IFILEDIALOG	276
8.11.1.3 PFD_OSX_ICON	276
8.12 FileDialog.h	276
8.13 include/core/LinkedList.cpp File Reference	298
8.14 LinkedList.cpp	298
8.15 include/core/LinkedList.hpp File Reference	303
8.16 LinkedList.hpp	303
8.17 include/core/Random.h File Reference	304
8.18 Random.h	305
8.19 include/core/Vector.h File Reference	305
8.20 Vector.h	305

8.21 include/draw/Arrow.cpp File Reference . . . . .	308
8.22 Arrow.cpp . . . . .	308
8.23 include/draw/Arrow.hpp File Reference . . . . .	310
8.24 Arrow.hpp . . . . .	310
8.25 include/draw/BackArrow.cpp File Reference . . . . .	310
8.26 BackArrow.cpp . . . . .	311
8.27 include/draw/BackArrow.hpp File Reference . . . . .	312
8.28 BackArrow.hpp . . . . .	312
8.29 include/draw/BaseDraw.cpp File Reference . . . . .	313
8.30 BaseDraw.cpp . . . . .	313
8.31 include/draw/BaseDraw.hpp File Reference . . . . .	313
8.32 BaseDraw.hpp . . . . .	314
8.33 include/draw/NodeInfo.cpp File Reference . . . . .	314
8.34 NodeInfo.cpp . . . . .	314
8.35 include/draw/NodeInfo.hpp File Reference . . . . .	317
8.36 NodeInfo.hpp . . . . .	317
8.37 include/draw/SingleNode.cpp File Reference . . . . .	318
8.38 SingleNode.cpp . . . . .	318
8.39 include/draw/SingleNode.hpp File Reference . . . . .	319
8.40 SingleNode.hpp . . . . .	319
8.41 include/draw/Square.cpp File Reference . . . . .	320
8.42 Square.cpp . . . . .	320
8.43 include/draw/Square.hpp File Reference . . . . .	321
8.44 Square.hpp . . . . .	321
8.45 include/draw/SquareInfo.cpp File Reference . . . . .	322
8.46 SquareInfo.cpp . . . . .	322
8.47 include/draw/SquareInfo.hpp File Reference . . . . .	323
8.48 SquareInfo.hpp . . . . .	323
8.49 include/libScene/AllScenes.hpp File Reference . . . . .	324
8.50 AllScenes.hpp . . . . .	324
8.51 include/libScene/BaseScene.cpp File Reference . . . . .	324
8.52 BaseScene.cpp . . . . .	324
8.53 include/libScene/BaseScene.hpp File Reference . . . . .	325
8.54 BaseScene.hpp . . . . .	325
8.55 include/libScene/CLLScene.cpp File Reference . . . . .	325
8.56 CLLScene.cpp . . . . .	326
8.57 include/libScene/CLLScene.hpp File Reference . . . . .	332
8.58 CLLScene.hpp . . . . .	333
8.59 include/libScene/ControlMenu.cpp File Reference . . . . .	333
8.60 ControlMenu.cpp . . . . .	333
8.61 include/libScene/ControlMenu.hpp File Reference . . . . .	334
8.62 ControlMenu.hpp . . . . .	335

8.63 include/libScene/DLLScene.cpp File Reference . . . . .	335
8.64 DLLScene.cpp . . . . .	336
8.65 include/libScene/DLLScene.hpp File Reference . . . . .	345
8.66 DLLScene.hpp . . . . .	345
8.67 include/libScene/DynamicArrayScene.cpp File Reference . . . . .	346
8.68 DynamicArrayScene.cpp . . . . .	346
8.69 include/libScene/DynamicArrayScene.hpp File Reference . . . . .	352
8.70 DynamicArrayScene.hpp . . . . .	352
8.71 include/libScene/Highlighter.cpp File Reference . . . . .	352
8.72 Highlighter.cpp . . . . .	353
8.73 include/libScene/Highlighter.hpp File Reference . . . . .	353
8.74 Highlighter.hpp . . . . .	354
8.75 include/libScene/MainMenu.cpp File Reference . . . . .	354
8.76 MainMenu.cpp . . . . .	354
8.77 include/libScene/MainMenu.hpp File Reference . . . . .	354
8.78 MainMenu.hpp . . . . .	355
8.79 include/libScene/MenuArray.cpp File Reference . . . . .	355
8.80 MenuArray.cpp . . . . .	355
8.81 include/libScene/MenuArray.hpp File Reference . . . . .	361
8.82 MenuArray.hpp . . . . .	361
8.83 include/libScene/MenuDataStructure.cpp File Reference . . . . .	362
8.84 MenuDataStructure.cpp . . . . .	362
8.85 include/libScene/MenuDataStructure.hpp File Reference . . . . .	365
8.86 MenuDataStructure.hpp . . . . .	366
8.87 include/libScene/MenuLinkedList.cpp File Reference . . . . .	367
8.88 MenuLinkedList.cpp . . . . .	367
8.89 include/libScene/MenuLinkedList.hpp File Reference . . . . .	372
8.90 MenuLinkedList.hpp . . . . .	372
8.91 include/libScene/QueueScene.cpp File Reference . . . . .	373
8.92 QueueScene.cpp . . . . .	373
8.93 include/libScene/QueueScene.hpp File Reference . . . . .	378
8.94 QueueScene.hpp . . . . .	378
8.95 include/libScene/SLLScene.cpp File Reference . . . . .	379
8.96 SLLScene.cpp . . . . .	379
8.97 include/libScene/SLLScene.hpp File Reference . . . . .	385
8.98 SLLScene.hpp . . . . .	386
8.99 include/libScene/StackScene.cpp File Reference . . . . .	386
8.100 StackScene.cpp . . . . .	386
8.101 include/libScene/StackScene.hpp File Reference . . . . .	391
8.102 StackScene.hpp . . . . .	391
8.103 include/libScene/StaticArrayScene.cpp File Reference . . . . .	392
8.104 StaticArrayScene.cpp . . . . .	392



8.105 include/libScene/StaticArrayScene.hpp File Reference	396
8.106 StaticArrayScene.hpp	396
8.107 include/MousePosition.cpp File Reference	396
8.108 MousePosition.cpp	397
8.109 include/MousePosition.hpp File Reference	397
8.110 MousePosition.hpp	397
8.111 include/stuff/button.cpp File Reference	397
8.112 button.cpp	398
8.113 include/stuff/button.hpp File Reference	399
8.114 button.hpp	399
8.115 include/stuff/CustomTextbox.cpp File Reference	400
8.116 CustomTextbox.cpp	400
8.117 include/stuff/CustomTextbox.hpp File Reference	401
8.118 CustomTextbox.hpp	401
8.119 include/stuff/RoundedRectangleShape.cpp File Reference	402
8.120 RoundedRectangleShape.cpp	402
8.121 include/stuff/RoundedRectangleShape.hpp File Reference	403
8.122 RoundedRectangleShape.hpp	404
8.123 include/stuff/Textbox.cpp File Reference	404
8.124 Textbox.cpp	404
8.125 include/stuff/Textbox.hpp File Reference	406
8.126 Textbox.hpp	406
8.127 include/stuff/ToStringWithPrecision.hpp File Reference	407
8.127.1 Function Documentation	407
8.127.1.1 to_string_with_precision()	407
8.128 ToStringWithPrecision.hpp	407
8.129 include/Window.cpp File Reference	407
8.130 Window.cpp	408
8.131 include/Window.hpp File Reference	410
8.132 Window.hpp	410
8.133 main.cpp File Reference	410
8.133.1 Function Documentation	411
8.133.1.1 main()	411
8.134 main.cpp	411
8.135 README.md File Reference	411

## Index

413



## Chapter 1

# Visualgo\_CS162



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

constants	11
constants::Arrow	13
constants::ControlMenu	13
constants::Highlighter	15
constants::Highlighter::DLL	15
constants::Highlighter::SLL	16
constants::LinkedList	16
constants::MenuArray	16
constants::MenuArray::AddMode	18
constants::MenuArray::AllocateMode	20
constants::MenuArray::CreateMode	21
constants::MenuArray::DeleteMode	23
constants::MenuArray::SearchMode	24
constants::MenuArray::UpdateMode	25
constants::MenuDataStructure	27
constants::MenuDataStructure::CreateMode	28
constants::MenuDataStructure::PushMode	30
constants::MenuLinkedList	31
constants::MenuLinkedList::AddMode	33
constants::MenuLinkedList::CreateMode	34
constants::MenuLinkedList::DeleteMode	36
constants::MenuLinkedList::SearchMode	37
constants::MenuLinkedList::UpdateMode	38
constants::NodeInfo	40
constants::sceneVariables	40
constants::Square	41
constants::TitleNode	41
pfd	41
pfd::internal	45
Random	45
sf	45



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

BaseDraw . . . . .	66
Array . . . . .	47
Arrow . . . . .	56
BackArrow . . . . .	62
NodeInfo . . . . .	173
SingleNode . . . . .	212
Square . . . . .	224
SquareInfo . . . . .	228
BaseScene . . . . .	67
CLLScene . . . . .	73
DLLScene . . . . .	93
DynamicArrayScene . . . . .	105
MainMenu . . . . .	141
QueueScene . . . . .	190
SLLScene . . . . .	215
StackScene . . . . .	232
StaticArrayScene . . . . .	239
Button . . . . .	71
ControlMenu . . . . .	82
CustomTextbox . . . . .	86
EventAnimation . . . . .	114
EventSquare . . . . .	118
pfd::internal::executor . . . . .	119
Highlighter . . . . .	129
LinkedList . . . . .	131
MenuArray . . . . .	143
MenuDataStructure . . . . .	148
MenuLinkedList . . . . .	151
MousePosition . . . . .	171
Window . . . . .	255
pfd::internal::platform . . . . .	190
pfd::internal::dialog . . . . .	89
pfd::internal::file_dialog . . . . .	123
pfd::open_file . . . . .	185

pfd::save_file . . . . .	202
pfd::select_folder . . . . .	205
pfd::message . . . . .	166
pfd::notify . . . . .	182
pfd::path . . . . .	188
pfd::settings . . . . .	207
pfd::internal::dialog . . . . .	89
sf::Shape	
sf::RoundedRectangleShape . . . . .	197
TextBox . . . . .	246
Vector< T > . . . . .	248
Vector< NodeInfo * > . . . . .	248
Vector< SquareInfo * > . . . . .	248



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">Array</a>	47
<a href="#">Arrow</a>	56
<a href="#">BackArrow</a>	62
<a href="#">BaseDraw</a>	66
<a href="#">BaseScene</a>	67
<a href="#">Button</a>	71
<a href="#">CLLScene</a>	73
<a href="#">ControlMenu</a>	82
<a href="#">CustomTextbox</a>	86
<a href="#">pfd::internal::dialog</a>	89
<a href="#">DLLScene</a>	93
<a href="#">DynamicArrayScene</a>	105
<a href="#">EventAnimation</a>	114
<a href="#">EventSquare</a>	118
<a href="#">pfd::internal::executor</a>	119
<a href="#">pfd::internal::file_dialog</a>	123
<a href="#">Highlighter</a>	129
<a href="#">LinkedList</a>	131
<a href="#">MainMenu</a>	141
<a href="#">MenuArray</a>	143
<a href="#">MenuDataStructure</a>	148
<a href="#">MenuLinkedList</a>	151
<a href="#">pfd::message</a>	166
<a href="#">MousePosition</a>	171
<a href="#">NodeInfo</a>	173
<a href="#">pfd::notify</a>	182
<a href="#">pfd::open_file</a>	185
<a href="#">pfd::path</a>	188
<a href="#">pfd::internal::platform</a>	190
<a href="#">QueueScene</a>	190
<a href="#">sf::RoundedRectangleShape</a>	
Specialized shape representing a rectangle with rounded corners	197
<a href="#">pfd::save_file</a>	202
<a href="#">pfd::select_folder</a>	205
<a href="#">pfd::settings</a>	207

<a href="#">SingleNode</a>	<a href="#">212</a>
<a href="#">SLLScene</a>	<a href="#">215</a>
<a href="#">Square</a>	<a href="#">224</a>
<a href="#">SquareInfo</a>	<a href="#">228</a>
<a href="#">StackScene</a>	<a href="#">232</a>
<a href="#">StaticArrayScene</a>	<a href="#">239</a>
<a href="#">TextBox</a>	<a href="#">246</a>
<a href="#">Vector&lt; T &gt;</a>	<a href="#">248</a>
<a href="#">Window</a>	<a href="#">255</a>

## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.cpp</a>	410
<a href="#">include/Constants.hpp</a>	259
<a href="#">include/MousePosition.cpp</a>	396
<a href="#">include/MousePosition.hpp</a>	397
<a href="#">include/Window.cpp</a>	407
<a href="#">include/Window.hpp</a>	410
<a href="#">include/core/Array.cpp</a>	267
<a href="#">include/core/Array.hpp</a>	271
<a href="#">include/core/EventAnimation.cpp</a>	272
<a href="#">include/core/EventAnimation.hpp</a>	273
<a href="#">include/core/FileDialog.h</a>	274
<a href="#">include/core/LinkedList.cpp</a>	298
<a href="#">include/core/LinkedList.hpp</a>	303
<a href="#">include/core/Random.h</a>	304
<a href="#">include/core/Vector.h</a>	305
<a href="#">include/draw/Arrow.cpp</a>	308
<a href="#">include/draw/Arrow.hpp</a>	310
<a href="#">include/draw/BackArrow.cpp</a>	310
<a href="#">include/draw/BackArrow.hpp</a>	312
<a href="#">include/draw/BaseDraw.cpp</a>	313
<a href="#">include/draw/BaseDraw.hpp</a>	313
<a href="#">include/draw/NodeInfo.cpp</a>	314
<a href="#">include/draw/NodeInfo.hpp</a>	317
<a href="#">include/draw/SingleNode.cpp</a>	318
<a href="#">include/draw/SingleNode.hpp</a>	319
<a href="#">include/draw/Square.cpp</a>	320
<a href="#">include/draw/Square.hpp</a>	321
<a href="#">include/draw/SquareInfo.cpp</a>	322
<a href="#">include/draw/SquareInfo.hpp</a>	323
<a href="#">include/libScene/AllScenes.hpp</a>	324
<a href="#">include/libScene/BaseScene.cpp</a>	324
<a href="#">include/libScene/BaseScene.hpp</a>	325
<a href="#">include/libScene/CLLScene.cpp</a>	325
<a href="#">include/libScene/CLLScene.hpp</a>	332
<a href="#">include/libScene/ControlMenu.cpp</a>	333

include/libScene/ControlMenu.hpp	334
include/libScene/DLLScene.cpp	335
include/libScene/DLLScene.hpp	345
include/libScene/DynamicArrayScene.cpp	346
include/libScene/DynamicArrayScene.hpp	352
include/libScene/Highlighter.cpp	352
include/libScene/Highlighter.hpp	353
include/libScene/MainMenu.cpp	354
include/libScene/MainMenu.hpp	354
include/libScene/MenuArray.cpp	355
include/libScene/MenuArray.hpp	361
include/libScene/MenuDataStructure.cpp	362
include/libScene/MenuDataStructure.hpp	365
include/libScene/MenuLinkedList.cpp	367
include/libScene/MenuLinkedList.hpp	372
include/libScene/QueueScene.cpp	373
include/libScene/QueueScene.hpp	378
include/libScene/SLLScene.cpp	379
include/libScene/SLLScene.hpp	385
include/libScene/StackScene.cpp	386
include/libScene/StackScene.hpp	391
include/libScene/StaticArrayScene.cpp	392
include/libScene/StaticArrayScene.hpp	396
include/stuff/button.cpp	397
include/stuff/button.hpp	399
include/stuff/CustomTextbox.cpp	400
include/stuff/CustomTextbox.hpp	401
include/stuff/RoundedRectangleShape.cpp	402
include/stuff/RoundedRectangleShape.hpp	403
include/stuff/Textbox.cpp	404
include/stuff/Textbox.hpp	406
include/stuff/ToStringWithPrecision.hpp	407

## 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

<a href="#">CREATE_BUTTON</a>	
<a href="#">ADD_BUTTON</a>	
<a href="#">DELETE_BUTTON</a>	
<a href="#">UPDATE_BUTTON</a>	
<a href="#">SEARCH_BUTTON</a>	
<a href="#">ALLOCATE_BUTTON</a>	
<a href="#">NONE</a>	

Definition at line 52 of file [Constants.hpp](#).

```
00052         {  
00053             CREATE\_BUTTON,  
00054             ADD\_BUTTON,  
00055             DELETE\_BUTTON,  
00056             UPDATE\_BUTTON,  
00057             SEARCH\_BUTTON,  
00058             ALLOCATE\_BUTTON,  
00059             NONE  
00060         };
```

#### 6.8.1.2 Type

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

##### Enumerator

<a href="#">DYNAMIC</a>	
<a href="#">STATIC</a>	

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](#).

```
00107         {
00108             POSITION_TEXTBOX,
00109             VALUE_TEXTBOX,
00110             NONE
00111         };
```

## 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

<a href="#">VALUE_TEXTBOX</a>	
<a href="#">NONE</a>	

Definition at line [151](#) of file [Constants.hpp](#).

```
00151     {
00152         VALUE_TEXTBOX,
00153         NONE
00154     };
```

### 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](#).

```
00073      {
00074          RANDOM_BUTTON,
00075          DEFINED_LIST_BUTTON,
00076          FILE_BUTTON,
00077          NONE
00078      };
```

#### 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

<a href="#">POSITION_TEXTBOX</a>	
<a href="#">NONE</a>	

Definition at line 117 of file [Constants.hpp](#).

```
00117 {
00118     POSITION_TEXTBOX,
00119     NONE
00120 };
```

### 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](#).

```
00142         {
00143             VALUE_TEXTBOX,
00144             NONE
00145         };
```

## 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](#).

```
00132         {
00133             POSITION_TEXTBOX,
00134             VALUE_TEXTBOX,
00135             NONE
00136         };
```

**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](#).

```
00160     {  
00161         CREATE_BUTTON,  
00162         PUSH_BUTTON,  
00163         POP_BUTTON,  
00164         CLEAR_BUTTON,  
00165         NONE  
00166     };
```

### 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](#).

```
00177     {
00178         RANDOM_BUTTON,
00179         DEFINED_LIST_BUTTON,
00180         FILE_BUTTON,
00181         NONE
00182     };
```

## 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](#).

```
00205     {  
00206         VALUE_TEXTBOX,  
00207         NONE  
00208     };
```

## 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](#).

```
00214         {
00215             CREATE_BUTTON,
00216             ADD_BUTTON,
00217             DELETE_BUTTON,
00218             UPDATE_BUTTON,
00219             SEARCH_BUTTON,
00220             NONE
00221         };
```

### 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](#).

```
00267      {
00268          POSITION_TEXTBOX,
00269          VALUE_TEXTBOX,
00270          NONE
00271      };
```

## 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](#).

```
00277         {
00278             POSITION_TEXTBOX,
00279             NONE
00280         };
```

## 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](#).

```
00302     {
00303         VALUE_TEXTBOX,
00304         NONE
00305     };
```

**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](#).

```
00292         {
00293             POSITION_TEXTBOX,
00294             VALUE_TEXTBOX,
00295             NONE
00296         };
```

## 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\\_SCENE](#),  
[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,
```

```
00078         no,  
00079         abort,  
00080         retry,  
00081         ignore,  
00082     };
```

### 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](#).

```
00085     {  
00086         ok = 0,  
00087         ok_cancel,  
00088         yes_no,  
00089         yes_no_cancel,  
00090         retry_cancel,  
00091         abort_retry_ignore,  
00092     };
```

### 6.28.1.3 icon

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

#### Enumerator

info	
warning	
error	
question	

Definition at line 94 of file [FileDialog.h](#).

```
00095     {  
00096         info = 0,  
00097         warning,  
00098         error,  
00099         question,  
00100     };
```

### 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](#).

```
00104     {
00105         none = 0,
00106         // For file open, allow multiselect.
00107         multiselect = 0x1,
00108         // For file save, force overwrite and disable the confirmation dialog.
00109         force_overwrite = 0x2,
00110         // For folder select, force path to be the provided argument instead
00111         // of the last opened directory, which is the Microsoft-recommended,
00112         // user-friendly behaviour.
00113         force_path = 0x4,
00114     };
```

## 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](#).

```
00117 { 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](#).

```
01025     {
01026         int not_first = 0;
01027         for (auto &e : v)
01028             s << (not_first++ ? " " : "") << e;
01029         return s;
01030     }
```

### 6.28.2.3 operator" | ()

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

Definition at line 116 of file [FileDialog.h](#).

```
00116 { 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.*





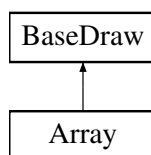
## Chapter 7

# 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](#).

```
00017 {
00018     DYNAMIC,
00019     STATIC
00020 };
```

### 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](#).

```
00009                                     : BaseDraw(window) {
00010     this->init(typeArray);
00011     this->createArray(0);
00012 }
```

**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](#).

```
00014                                     : BaseDraw(window) {
00015     this->init(typeArray);
00016     this->createArray(size);
00017 }
```

**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](#).

```
00019                                     :
00020     BaseDraw(window) {
00021     this->init(typeArray);
00022     this->createArray(std::move(values));
00023 }
```

**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](#).

```
00234                                     {
00235     if (position < 0 || position > this->size)
00236         return;
00237
00238     ++this->size;
00239     if (this->typeArray == TypeArray::DYNAMIC && this->size > this->getSquaresSize()) {
00240         this->squares.push_back(new SquareInfo(
00241             this->window,
00242             "",
00243             sf::Vector2f(
00244                 constants::Square::originNode.x + static_cast<float>(this->getSquaresSize()) *
constants::Square::offsetX,
00245                 constants::Square::originNode.y
00246             )
00247         ));
00248         this->squaresTemp.resize(this->size);
00249         for (int i = 0; i < this->size; ++i) {
00250             this->squaresTemp[i] = new SquareInfo(
00251                 this->window,
00252                 "",
00253                 sf::Vector2f(
00254                     constants::Square::originNode.x + static_cast<float>(i) *
constants::Square::offsetX,
00255                     constants::Square::originNode.y + constants::Square::offsetY
00256                 )
00257             );
00258             this->squaresTemp[i]->setValue(this->squares[i]->getValue());
00259         }
00260     }
00261
00262     if (size > this->getSquaresSize())
00263         --this->size;
00264
00265     for (int i = this->size - 1; i > position; --i)
00266         this->squares[i]->setValue(this->squares[i - 1]->getValue());
00267     this->squares[position]->setValue(std::move(value));
00268     for (int i = 0; i < position; ++i)
00269         this->squares[i]->setValue(this->squares[i]->getValue());
00270
00271     this->currentEvent = 0;
00272     this->events = listEvents;
00273 }
```

### 7.1.4.2 allocateSquare()

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

Definition at line 203 of file [Array.cpp](#).

```
00203                                     {
00204     this->squaresTemp.resize(_size);
00205     // this->squares.resize();
00206
00207     while (this->squares.size() < _size)
00208         this->squares.push_back(new SquareInfo(
00209             this->window,
00210             "",
00211             sf::Vector2f(
00212                 constants::Square::originNode.x + static_cast<float>(this->squares.size()) *
constants::Square::offsetX,
00213                 constants::Square::originNode.y
00214             )
00215         ));
00216 }
```

```

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](#).

```
00275                                     {
00276     if (position < 0 || position >= this->size)
00277         return;
00278
00279     --this->size;
00280
00281     for (int i = position; i < this->size; ++i)
00282         this->squares[i]->setValue(this->squares[i + 1]->getValue());
00283     for (int i = 0; i < position; ++i)
00284         this->squares[i]->setValue(this->squares[i]->getValue());
00285     this->squares[this->size]->setValue(this->squares[this->size]->getValue());
00286
00287     this->currentEvent = 0;
00288     this->events = listEvents;
00289 }
```

#### 7.1.4.6 findValue()

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

Definition at line 55 of file [Array.cpp](#).

```
00055                                     {
00056     for (int i = 0; i < this->size; i++) {
00057         if (this->squares[i]->getValue() == value)
00058             return i;
00059     }
00060     return this->size;
00061 }
```

#### 7.1.4.7 getSize()

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

Definition at line 63 of file [Array.cpp](#).

```
00063     {
00064     return this->size;
00065 }
```

#### 7.1.4.8 getSquaresSize()

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

Definition at line 199 of file [Array.cpp](#).

```
00199     {
00200     return (int)this->squares.size();
00201 }
```

#### 7.1.4.9 init()

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

Definition at line 24 of file [Array.cpp](#).

```
00024 {
00025     this->typeArray = typeArray;
00026     this->highlighter = nullptr;
00027     this->delayTime = constants::LinkedList::DELAY_TIME;
00028     this->size = 0;
00029 }
```

#### 7.1.4.10 initHighlighter()

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

Definition at line 92 of file [Array.cpp](#).

```
00092 {
00093     delete this->highlighter;
00094     this->highlighter = new Highlighter(
00095         this->window,
00096         linesCount,
00097         codePath
00098     );
00099 }
```

#### 7.1.4.11 processControlMenu()

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

Definition at line 67 of file [Array.cpp](#).

```
00067 {
00068     if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)
00069         return;
00070     switch (status){
00071         case ControlMenu::StatusCode::PREVIOUS:
00072             if (this->currentEvent > 0)
00073                 --this->currentEvent;
00074             break;
00075         case ControlMenu::StatusCode::PAUSE:
00076             // std::cout << "PAUSE" << std::endl;
00077             break;
00078         case ControlMenu::StatusCode::PLAY:
00079             if (this->currentEvent + 1 < this->events.size()) {
00080                 this->isDelay = true;
00081                 this->clock.restart();
00082             }
00083         case ControlMenu::StatusCode::NEXT:
00084             if (this->currentEvent + 1 < this->events.size())
00085                 ++this->currentEvent;
00086             break;
00087         default:
00088             break;
00089     }
00090 }
```

#### 7.1.4.12 render()

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

Implements [BaseDraw](#).

Definition at line 31 of file [Array.cpp](#).

```
00031     {
00032         for (auto &square : this->squares) {
00033             square->render();
00034         }
00035         for (auto &square: this->squaresTemp) {
00036             square->render();
00037         }
00038     }
```

#### 7.1.4.13 renderHighlighter()

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

Definition at line 40 of file [Array.cpp](#).

```
00040     {
00041         if (this->highlighter)
00042             this->highlighter->render();
00043     }
```

#### 7.1.4.14 resetEvents()

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

Definition at line 139 of file [Array.cpp](#).

```
00139     {
00140         delete this->highlighter;
00141         this->highlighter = nullptr;
00142         this->currentEvent = 0;
00143         this->events.clear();
00144         this->squaresTemp.clear();
00145
00146         while (!this->squares.empty() && this->squares.back()->getStatus() == Square::Status::hidden)
00147             this->squares.pop_back();
00148
00149         for (int i = 0; i < this->size; ++i)
00150             this->squares[i]->setStatus(Square::Status::active);
00151         for (int i = this->size; i < this->squares.size(); ++i)
00152             this->squares[i]->setStatus(Square::Status::inactive);
00153         if (this->size)
00154             this->squares[this->size - 1]->setTitle("n");
00155     }
```

#### 7.1.4.15 searchSquare()

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

Definition at line 301 of file [Array.cpp](#).

```
00301     {
00302         this->currentEvent = 0;
00303         this->events = listEvents;
00304     }
```



**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 }

```

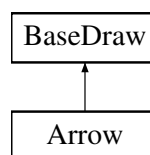
The documentation for this class was generated from the following files:

- [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](#).

```
00007                                     : BaseDraw(window) {
00008     this->points[0] = start;
00009     this->points[1] = end;
00010
00011     this->arrowTexture[0].loadFromFile("../assets/arrow/arrow_black.png");
00012     this->arrowTexture[1].loadFromFile("../assets/arrow/arrow_orange.png");
00013
00014     this->arrowTexture[0].setSmooth(true);
00015     this->arrowTexture[1].setSmooth(true);
00016
00017     this->arrowSprite.setTexture(this->arrowTexture[0]);
00018     sf::Vector2i topLeftCorner(
00019         static_cast<int>(this->arrowTexture[0].getSize().x / 2.0 - constants::Arrow::sizeArrow.x /
00020 2.0),
00021         static_cast<int>(this->arrowTexture[0].getSize().y / 2.0 - constants::Arrow::sizeArrow.y /
00022 2.0)
00023     );
00024     this->arrowSprite.setTextureRect(sf::IntRect(
00025         topLeftCorner.x,
00026         topLeftCorner.y,
00027         constants::Arrow::sizeArrow.x,
00028         constants::Arrow::sizeArrow.y
00029     ));
00030     this->autoScale();
00031     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 -
00036 //              constants::Arrow::sizeRectangle.x / 2.0),
00037 //          static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
00038 //              constants::Arrow::sizeRectangle.y / 2.0)
00039 //      );
00040 //      this->rectangleSprite.setTexture(this->rectangleTexture[0]);
00041 //      this->rectangleSprite.setTextureRect(sf::IntRect(
00042 //          topLeftCorner.x,
00043 //          topLeftCorner.y,
00044 //          constants::Arrow::sizeRectangle.x,
00045 //          constants::Arrow::sizeRectangle.y
00046 //      ));
00047 //      this->rectangleSprite.setScale(
00048 //          constants::Arrow::defaultScaleRectangle.x,
00049 //          constants::Arrow::defaultScaleRectangle.y
00050 //      );
00051 //      this->rectangleSprite.setOrigin(
00052 //          0,
00053 //          this->rectangleSprite.getLocalBounds().height / 2.0f
00054 //      );
00055 //      this->rectangleSprite.setPosition(sf::Vector2f(50, 200));
00056 //      this->rectangleSprite.setRotation(angle);
00057
00058 this->hasSetMid = false;
00059 }

```

## 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 -
00098                 this->points[0].y, 2)
00099             ) - constants::NodeInfo::radius - 2.f
00100         );
00101     this->arrowSprite.setScale(
00102         this->length / this->arrowSprite.getLocalBounds().width,
00103         constants::Arrow::defaultScaleArrow.y
00104     );
00105     this->arrowSprite.setOrigin(
00106         0,
00107         this->arrowSprite.getLocalBounds().height / 2.0f
00108     );
00109     this->arrowSprite.setPosition(this->points[0]);
00110 }

```

### 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::setPosition (
    sf::Vector2f start,
    sf::Vector2f end,
    bool needSetMid )
```

Definition at line 74 of file [Arrow.cpp](#).

```
00074 {
00075     this->points[0] = start;
00076     this->points[1] = end;
00077     if (needSetMid) {
00078         this->hasSetMid = false;
00079         this->setMid();
00080     }
00081     else {
00082         this->arrowSprite.setPosition(this->points[0]);
00083         this->autoScale();
00084         this->autoRotate();
00085     }
00086 }
```

### 7.2.3.8 setStart()

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

Definition at line 121 of file [Arrow.cpp](#).

```
00121 {
00122     this->setPositions(start, this->points[1], needSetMid);
00123 }
```

### 7.2.3.9 show()

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

Definition at line 131 of file [Arrow.cpp](#).

```
00131 {
00132     sf::Color tmp = this->arrowSprite.getColor();
00133     tmp.a = 255;
00134     this->arrowSprite.setColor(tmp);
00135 }
```

### 7.2.3.10 toggleActiveColor()

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

Definition at line 64 of file [Arrow.cpp](#).

```
00064 {
00065     this->arrowSprite.setTexture(this->arrowTexture[1]);
00066     // this->rectangleSprite.setTexture(this->rectangleTexture[1]);
00067 }
```

## 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](#).

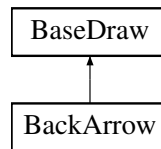
The documentation for this class was generated from the following files:

- [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 -
00030         constants::Arrow::sizeRectangle.x / 2.0),
00031         static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
00032         constants::Arrow::sizeRectangle.y / 2.0)
00033     );
00034     for (auto & rectangleSprite : this->rectangleSprites) {
00035         rectangleSprite.setTexture(this->rectangleTexture[0]);
00036         rectangleSprite.setTextureRect(sf::IntRect(
00037             topLeftCorner.x,
00038             topLeftCorner.y,
00039             constants::Arrow::sizeRectangle.x,
00040             constants::Arrow::sizeRectangle.y
00041         ));
00042     }
00043     this->setPosition(start, end);
```

## 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](#).

```
00093     {
00094         float length = sqrtf(
00095             powf(this->points[3].x - this->points[2].x, 2) + powf(this->points[3].y -
00096                 this->points[2].y, 2)
00097             );
00098         this->rectangleSprites[0].setScale(
00099             length / this->rectangleSprites[0].getLocalBounds().width,
00100             constants::Arrow::defaultScaleRectangle.y
00101         );
00102         length = sqrtf(
00103             powf(this->points[3].x - this->points[1].x, 2) + powf(this->points[3].y -
00104                 this->points[1].y, 2)
00105             );
00106         this->rectangleSprites[1].setScale(
00107             length / this->rectangleSprites[1].getLocalBounds().width,
00108             constants::Arrow::defaultScaleRectangle.y
00109         );
00110         this->rectangleSprites[0].setOrigin(
00111             this->rectangleSprites[0].getLocalBounds().width / 2.0f,
00112             0
00113         );
00114         this->rectangleSprites[1].setOrigin(
00115             this->rectangleSprites[1].getLocalBounds().width,
00116             this->rectangleSprites[1].getLocalBounds().height / 2.0f
00117         );
00118         this->rectangleSprites[0].setPosition(
00119             (this->points[3].x + this->points[2].x) / 2.0f,
00120             (this->points[3].y + this->points[2].y) / 2.0f
00121         );
00122         this->rectangleSprites[1].setPosition(this->points[1]);
00123     }
```

### 7.3.3.3 hide()

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

Definition at line 57 of file [BackArrow.cpp](#).

```
00057     {
00058         this->isShow = false;
00059     }
```

### 7.3.3.4 render()

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

Implements [BaseDraw](#).

Definition at line 45 of file [BackArrow.cpp](#).

```
00045     {
00046         if (this->isShow) {
00047             this->window->draw(this->rectangleSprites[0]);
00048             this->window->draw(this->rectangleSprites[1]);
00049             this->arrow->render();
00050         }
00051     }
```

### 7.3.3.5 resetColor()

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

Definition at line 67 of file [BackArrow.cpp](#).

```
00067      {
00068          this->rectangleSprites[0].setTexture(this->rectangleTexture[0]);
00069          this->rectangleSprites[1].setTexture(this->rectangleTexture[0]);
00070          this->arrow->resetColor();
00071      }
```

### 7.3.3.6 setPosition()

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

Definition at line 73 of file [BackArrow.cpp](#).

```
00073      {
00074          this->points[0] = end;
00075          this->points[1] = start;
00076          if (end == start) {
00077              this->hide();
00078              return;
00079          }
00080          this->points[2] = sf::Vector2f(
00081              this->points[0].x,
00082              this->points[0].y - constants::NodeInfo::offsetX
00083          );
00084          this->points[3] = sf::Vector2f(
00085              this->points[1].x,
00086              this->points[2].y
00087          );
00088          this->arrow->setPositions(this->points[2], this->points[0], false);
00089          this->autoRotate();
00090          this->autoScale();
00091      }
```

### 7.3.3.7 show()

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

Definition at line 53 of file [BackArrow.cpp](#).

```
00053      {
00054          this->isShow = true;
00055      }
```

### 7.3.3.8 toggleActiveColorNode()

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

Definition at line 61 of file [BackArrow.cpp](#).

```
00061      {
00062          this->rectangleSprites[0].setTexture(this->rectangleTexture[1]);
00063          this->rectangleSprites[1].setTexture(this->rectangleTexture[1]);
00064          this->arrow->toggleActiveColor();
00065      }
```

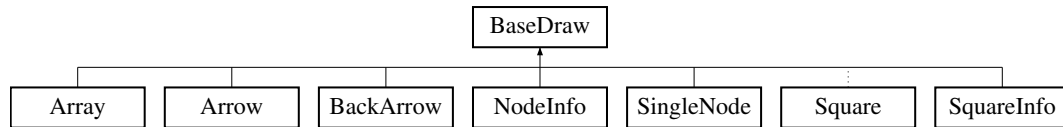
The documentation for this class was generated from the following files:

- [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](#).

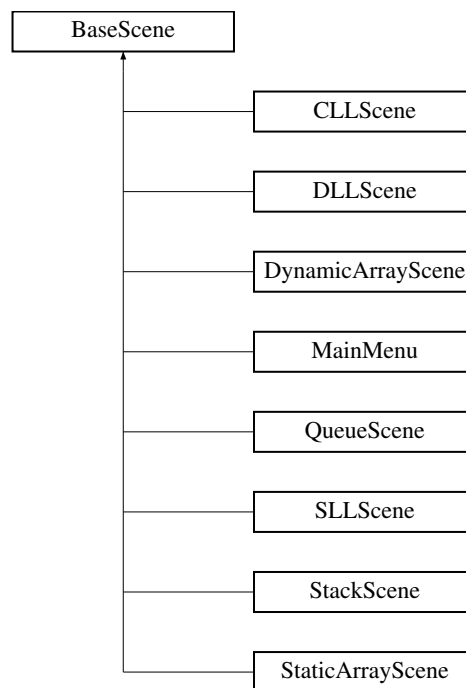
The documentation for this class was generated from the following files:

- [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](#).

```
00026                                     {
00027     this->setWindow(window);
00028     this->isMenuOpen = false;
00029     this->isDemoCodeOpen = false;
00030
00031     this->controlMenu = new ControlMenu(this->window);
00032 }
```

### 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](#).

```
00011                                     {
00012     this->modeButton = new Button(
00013         this->window,
00014         position,
00015         constants::modeButtonSize,
00016         textString,
00017         textString,
00018         constants::sizeTextModeButton,
00019         sf::Color::Black,
00020         constants::normalGray,
00021         constants::hoverGray,
00022         constants::clickGray
00023     );
00024 }
```

### 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](#).

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

#### 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 ( )
```

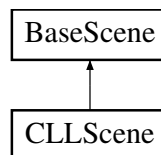
The documentation for this class was generated from the following file:

- 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](#).

```
00007                                     : BaseScene(window) {
00008     this->init();
00009 }
```

## 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     }

```

```

00293     this->linkedList->initHighlighter(
00294         constants::Highlighter::SLL::CODES_PATH[1].second,
00295         constants::Highlighter::SLL::CODES_PATH[1].first
00296     );
00297
00298     std::vector<EventAnimation> events;
00299     EventAnimation event;
00300
00301     if (!chosenNode) {
00302         event.titleNodes.emplace_back(chosenNode, "head|temp");
00303         event.colorNodes.push_back(chosenNode);
00304         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00305         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00306         event.lines = {0, 1};
00307
00308         events.emplace_back(event);
00309
00310         if (this->linkedList->getSize() > 1) {
00311             event.reset();
00312             event.titleNodes = {
00313                 {chosenNode, "temp"},
00314                 {1, "head"}
00315             };
00316             event.colorNodes.push_back(1);
00317             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00318             event.isPrintNormal = true;
00319             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00320             event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
00321             event.lines = {2};
00322
00323             events.emplace_back(event);
00324         }
00325
00326         event.reset();
00327         event.titleNodes.emplace_back(1, "head");
00328         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00329         event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
00330         event.lines = {3};
00331
00332         events.emplace_back(event);
00333     } else {
00334         event.reset();
00335         event.titleNodes.emplace_back(0, "head|current");
00336         event.colorNodes.push_back(0);
00337         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00338         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00339         event.lines = {5};
00340
00341         events.emplace_back(event);
00342
00343         for (int i = 0; i < chosenNode; ++i) {
00344             event.reset();
00345             event.titleNodes = {
00346                 {0, "head"},
00347                 {i, "current"}
00348             };
00349             event.colorNodes.push_back(i);
00350             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00351             event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00352             event.lines = {6};
00353
00354             events.emplace_back(event);
00355
00356             if (i == chosenNode - 1) break;
00357
00358             event.reset();
00359             event.titleNodes = {
00360                 {0, "head"},
00361                 {i, "current"}
00362             };
00363             event.colorNodes.push_back(i);
00364             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00365             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00366             event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00367             event.lines = {7};
00368
00369             events.emplace_back(event);
00370         }
00371
00372         event.reset();
00373         event.titleNodes = {
00374             {0, "head"},
00375             {chosenNode, "temp"},
00376             {chosenNode - 1, "current"}
00377         };
00378         event.colorNodes.push_back(chosenNode);
00379         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);

```

```

00380     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00381     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00382     event.lines = {8};
00383
00384     events.emplace_back(event);
00385
00386     if (chosenNode != this->linkedList->getSize() - 1) {
00387         event.reset();
00388         event.titleNodes = {
00389             {0, "head"},
00390             {chosenNode, "temp"},
00391             {chosenNode - 1, "current"}
00392         };
00393         event.colorNodes.push_back(chosenNode);
00394         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00395         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00396         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00397         event.isPrintNormal = true;
00398         event.lines = {9};
00399
00400         events.emplace_back(event);
00401
00402         event.reset();
00403         event.titleNodes.emplace_back(0, "head");
00404         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00405         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00406         event.lines = {10};
00407
00408         events.emplace_back(event);
00409     } else {
00410         event.reset();
00411         event.titleNodes = {
00412             {0, "head"},
00413             {chosenNode, "temp"},
00414             {chosenNode - 1, "current"}
00415         };
00416         event.colorNodes.push_back(chosenNode);
00417         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00418         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00419         event.indexBackArrow = {chosenNode - 1, 0};
00420         event.lines = {9};
00421
00422         events.emplace_back(event);
00423
00424         event.reset();
00425         event.titleNodes.emplace_back(0, "head");
00426         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00427         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00428         event.indexBackArrow = {chosenNode - 1, 0};
00429         event.lines = {10};
00430
00431         events.emplace_back(event);
00432     }
00433 }
00434
00435 return events;
00436 }

```

### 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](#).

```

00011                                     {
00012     if (this->isMenuOpen)
00013         this->menu->pollEvents(event, mousePosView);
00014
00015     this->controlMenu->pollEvents(event, mousePosView);
00016 }

```



### 7.7.3.4 render()

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

Implements [BaseScene](#).

Definition at line 123 of file [CLLScene.cpp](#).

```
00123     {
00124     if (this->isMenuOpen)
00125         this->menu->render();
00126
00127     if (this->isDemoCodeOpen)
00128         this->linkedList->renderHighlighter();
00129
00130     this->controlMenu->render();
00131     this->linkedList->render();
00132 }
```

### 7.7.3.5 reset()

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

Definition at line 139 of file [CLLScene.cpp](#).

```
00139     {
00140     this->menu->resetActiveOptionsMenu();
00141 }
```

### 7.7.3.6 searchModeEvents()

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

Definition at line 506 of file [CLLScene.cpp](#).

```
00506     {
00507     this->linkedList->resetEvents();
00508     this->linkedList->initHighlighter (
00509         constants::Highlighter::SLL::CODES_PATH[3].second,
00510         constants::Highlighter::SLL::CODES_PATH[3].first
00511     );
00512
00513     std::vector<EventAnimation> events;
00514     EventAnimation event;
00515
00516     event.titleNodes.emplace_back(0, "head|current");
00517     event.colorNodes.push_back(0);
00518     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00519     event.lines = {0};
00520
00521     events.emplace_back(event);
00522
00523     for (int i = 0; i <= chosenNode; ++i) {
00524         if (i == chosenNode && chosenNode == this->linkedList->getSize())
00525             break;
00526
00527         event.reset();
00528         event.titleNodes = {
00529             {0, "head"},
00530             {i, "current"}
00531         };
00532         event.colorNodes.push_back(i);
00533         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00534         event.lines = {1};
00535
00536         events.emplace_back(event);
00537
00538         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->getActiveOptionsMenu();
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 ==
00035 constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
00036                         if (this->menu->createModeValue[1] == "None")
00037                             break;
00038                         std::vector<std::string> values;
00039                         std::string value = this->menu->createModeValue[1];
00040                         std::stringstream ss(value);
00041                         std::string token;
00042                         while (std::getline(ss, token, ',')) {
00043                             values.push_back(token);
00044                         }
00045                         this->linkedList->createLinkedList(values);
00046                     } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
00047                         if (this->menu->createModeValue[2] == "None")
00048                             break;
00049                         std::vector<std::string> values;
00050                         std::string value = this->menu->createModeValue[2];
00051                         std::stringstream ss(value);

```

```

00051         std::string token;
00052         while (std::getline(ss, token, ','))
00053             values.push_back(token);
00054         this->linkedList->createLinkedList(values);
00055         this->menu->createModeValue[2] = "None";
00056     }
00057     this->controlMenu->reset();
00058     break;
00059     case constants::MenuLinkedList::Button::ADD_BUTTON:
00060         if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
this->menu->addModeValue[0].empty())
00061             break;
00062
00063         this->linkedList->addNode(
00064             std::stoi(this->menu->addModeValue[0]),
00065             this->menu->addModeValue[1],
00066             this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00067         );
00068
00069         std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
<< std::endl;
00070         this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00071         this->controlMenu->reset();
00072         break;
00073     case constants::MenuLinkedList::Button::DELETE_BUTTON:
00074         if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
00075             break;
00076
00077         this->linkedList->deleteNode(
00078             std::stoi(this->menu->deleteModeValue),
00079             this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
00080         );
00081
00082         std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00083         this->menu->deleteModeValue = "None";
00084         this->controlMenu->reset();
00085         break;
00086     case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00087         if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
00088             break;
00089
00090         this->linkedList->updateNode(
00091             std::stoi(this->menu->updateModeValue[0]),
00092             this->menu->updateModeValue[1],
00093             this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00094         );
00095
00096         std::cout << "Update: " << this->menu->updateModeValue[0] << " " <<
this->menu->updateModeValue[1] << std::endl;
00097         this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00098         this->controlMenu->reset();
00099         break;
00100     case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00101         if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00102             break;
00103
00104         this->linkedList->searchNode(
00105             this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
00106         );
00107
00108         std::cout << "Search: " << this->menu->searchModeValue << std::endl;
00109         this->menu->searchModeValue = "None";
00110         this->controlMenu->reset();
00111         break;
00112     }
00113 }
00114
00115 this->controlMenu->update();
00116
00117 this->linkedList->processControlMenu(this->controlMenu->getStatus());
00118 this->linkedList->setSpeed(this->controlMenu->getSpeed());
00119
00120 this->linkedList->update();
00121 }

```

### 7.7.3.8 updateModeEvents()

```

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

```

Definition at line 438 of file [CLLScene.cpp](#).

```

00438                                     {
00439     this->linkedList->resetEvents();
00440     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00441         return {};
00442
00443     this->linkedList->initHighlighter(
00444         constants::Highlighter::SLL::CODES_PATH[2].second,
00445         constants::Highlighter::SLL::CODES_PATH[2].first
00446     );
00447
00448     std::vector<EventAnimation> events;
00449     EventAnimation event;
00450
00451     event.titleNodes.emplace_back(0, "head|current");
00452     event.colorNodes.push_back(0);
00453     event.isPrintPreVal = true;
00454     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00455     event.lines = {0};
00456
00457     events.emplace_back(event);
00458
00459     if (chosenNode) {
00460         for (int i = 0; i <= chosenNode; ++i) {
00461             event.reset();
00462             event.titleNodes = {
00463                 {0, "head"},
00464                 {i, "current"}
00465             };
00466             event.colorNodes.push_back(i);
00467             event.isPrintPreVal = true;
00468             event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00469             event.lines = {1};
00470
00471             events.emplace_back(event);
00472
00473             if (i == chosenNode) break;
00474
00475             event.reset();
00476             event.titleNodes = {
00477                 {0, "head"},
00478                 {i, "current"}
00479             };
00480             event.colorNodes.push_back(i);
00481             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00482             event.isPrintPreVal = true;
00483             event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00484             event.lines = {2};
00485
00486             events.emplace_back(event);
00487         }
00488     }
00489
00490     event.reset();
00491     if (chosenNode == 0)
00492         event.titleNodes.emplace_back(0, "head|current");
00493     else
00494         event.titleNodes = {
00495             {0, "head"},
00496             {chosenNode, "current"}
00497         };
00498     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00499     event.lines = {3};
00500
00501     events.emplace_back(event);
00502
00503     return events;
00504 }

```

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](#).

```
00023                                     {
00024     PREVIOUS,
00025     PAUSE,
00026     PLAY,
00027     NEXT,
00028     None
00029 } 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](#).

```
00007     {
00008         this->window = window;
00009
00010         for (int i = 0; i < constants::ControlMenu::BUTTON_COUNT; ++i) {
00011             buttons[i] = new Button(
00012                 this->window,
00013                 constants::ControlMenu::buttonPos[i],
00014                 constants::ControlMenu::buttonSize,
00015                 constants::ControlMenu::BUTTON_NAMES[i],
00016                 constants::ControlMenu::BUTTON_NAMES[i],
00017                 constants::ControlMenu::BUTTON_NAME_SIZE,
00018                 sf::Color::Black,
00019                 constants::normalGray,
00020                 constants::hoverGray,
00021                 constants::clickGray
00022             );
00023         }
00024
00025         this->font.loadFromFile(constants::fontPath);
00026         this->textSpeed.setFont(font);
00027         this->textSpeed.setString(to_string_with_precision(this->speed));
00028         this->textSpeed.setCharacterSize(constants::ControlMenu::TEXT_SIZE);
00029         this->textSpeed.setFillColor(sf::Color::Black);
00030         this->textSpeed.setOrigin(
00031             this->textSpeed.getLocalBounds().width / 2.0f,
00032             this->textSpeed.getLocalBounds().height / 2.0f
00033         );
00034         this->textSpeed.setPosition(
00035             constants::ControlMenu::buttonPos[3].x + constants::ControlMenu::buttonSize.x * 2,
00036             constants::ControlMenu::buttonPos[3].y + constants::ControlMenu::buttonSize.y / 2.0f
00037         );
00038
00039         this->status = StatusCode::None;
00040         this->speed = 1;
00041     }
```

### 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](#).

```
00096     {
00097         return this->speed;
00098     }
```

## 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](#).

```
00100      {
00101      this->status = StatusCode::None;
00102  }
```

#### 7.8.4.6 update()

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

Definition at line 75 of file [ControlMenu.cpp](#).

```
00075      {
00076      for (auto &button : buttons) {
00077          button->update();
00078      }
00079      this->textSpeed.setString(to_string_with_precision(this->speed));
00080  }
```

### 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](#).

```
00008 {
00009     this->window = window;
00010     this->titleString = std::move(_titleString);
00011     this->position = position;
00012
00013     font.loadFromFile(constants::fontPath);
00014     this->title.setFont(font);
00015     this->title.setCharacterSize(size);
00016     this->title.setFillColor(sf::Color::Black);
00017     this->title.setString(this->titleString);
00018     this->title.setPosition(this->position);
00019
00020     float width = this->title.findCharacterPos(this->titleString.size() - 1).x -
this->title.findCharacterPos(0).x;
00021
00022     //     std::cout << width << ' ' << this->title.getString().getSize() << std::endl;
00023
00024     this->maxLength = maxLength;
00025
00026     this->textbox = new TextBox(
00027         this->window,
00028         sf::Vector2f(this->position.x + width + 10, this->position.y),
00029         20,
00030         sf::Color::Black,
00031         sf::Color::White,
00032         this->maxLength
00033     );
00034
00035     this->goButton = new Button(
00036         this->window,
00037         sf::Vector2f(this->textbox->getBox().getPosition().x + this->textbox->getBox().getSize().x +
10, this->position.y),
00038         constants::goButtonSize,
00039         "Go",
00040         "Go",
00041         20,
00042         sf::Color::Black,
00043         constants::normalGray,
00044         constants::hoverGray,
00045         constants::clickGray
00046     );
00047
00048     this->isGoButtonClicked = false;
00049 }
```

### 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](#).

```
00070     {
00071         if (this->isGoButtonClicked) {
00072             this->isGoButtonClicked = false;
00073             return this->textbox->getTextString();
00074         }
00075         return "None";
00076     }
```

### 7.9.3.2 pollEvent()

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

Definition at line 51 of file [CustomTextbox.cpp](#).

```
00051     {
00052         this->textbox->pollEvent(event);
00053         if (this->goButton->pollEvent(mousePosView)) {
00054             this->isGoButtonClicked = true;
00055             //         std::cout << "Go button clicked!\n";
00056         }
00057     }
```

### 7.9.3.3 render()

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

Definition at line 64 of file [CustomTextbox.cpp](#).

```
00064     {
00065         this->window->draw(this->title);
00066         this->textbox->render();
00067         this->goButton->render();
00068     }
```

### 7.9.3.4 resetInput()

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

Definition at line 78 of file [CustomTextbox.cpp](#).

```
00078     {
00079         this->textbox->resetInput();
00080     }
```

### 7.9.3.5 update()

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

Definition at line 59 of file [CustomTextbox.cpp](#).

```
00059     {
00060         this->textbox->update();
00061         this->goButton->update();
00062     }
```

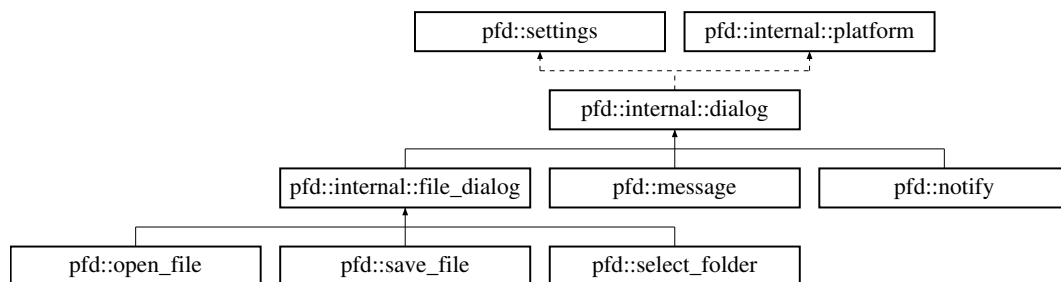
The documentation for this class was generated from the following files:

- [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](#).

```
00994     {
00995         switch (_choice)
00996         {
00997             case choice::ok_cancel: return "okcancel";
00998             case choice::yes_no: return "yesno";
00999             case choice::yes_no_cancel: return "yesnocancel";
01000             case choice::retry_cancel: return "retrycancel";
01001             case choice::abort_retry_ignore: return "abortretryignore";
01002             /* case choice::ok: */ default: return "ok";
01003         }
01004     }
```

### 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](#).

```
00981     {
00982         #if __APPLE__
00983             return { "osascript" };
00984         #else
00985             return { flags(flag::has_zenity) ? "zenity"
00986                     : flags(flag::has_matedialog) ? "matedialog"
00987                     : flags(flag::has_qarma) ?
00988                       "qarma"
00989                     :
00990                       flags(flag::has_kdialog) ? "kdialog"
00991                     : "echo" };
00992         #endif
00993     }
```

### 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](#).

```
01007     {
01008         switch (_icon)
01009         {
01010             case icon::warning: return "warning";
01011             case icon::error: return "error";
01012             case icon::question: return "question";
01013             // Zenity wants "information" but WinForms wants "info"
01014             /* case icon::info: */ default:
01015             #if _WIN32
01016                 return "info";
01017             #else
01018                 return "information";
01019             #endif
01020         }
01021     }
```

#### 7.10.3.4 kill()

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

Definition at line 970 of file [FileDialog.h](#).

```
00971 {  
00972     return m_async->kill();  
00973 }
```

#### 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](#).

```
01045 {  
01046     return "\"" + std::regex_replace(str, std::regex("[\\\\""]), "\\\"$&") + "\"";  
01047 }
```

#### 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](#).

```
01037 {  
01038     return "'" + std::regex_replace(str, std::regex("['\""]), "$&$&") + "'";  
01039 }
```

#### 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](#).

```
00966 {  
00967     return m_async->ready(timeout);  
00968 }
```

#### 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](#).

```
01052 {  
01053     return "\"" + std::regex_replace(str, std::regex("\""), "\\\"") + "\"";  
01054 }
```

## 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](#).

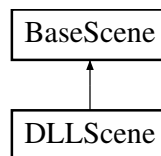
The documentation for this class was generated from the following file:

- [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](#).

```
00007                                     : BaseScene(window) {  
00008     this->init();  
00009 }
```

## 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         }
```

```

00221         else {
00222             event.titleNodes.emplace_back(chosenNode, "head|tail");
00223             event.lines = {5, 6, 7};
00224         }
00225         event.colorNodes = {chosenNode};
00226
00227         events.emplace_back(event);
00228     }
00229     else if (chosenNode == size) {
00230         this->linkedList->initHighlighter(
00231             constants::Highlighter::DLL::CODES_PATH[1].second,
00232             constants::Highlighter::DLL::CODES_PATH[1].first
00233         );
00234
00235         event.titleNodes.emplace_back(chosenNode, "temp");
00236         if (size == 1)
00237             event.titleNodes.emplace_back(0, "head|tail");
00238         else if (size > 1){
00239             event.titleNodes.emplace_back(0, "head");
00240             event.titleNodes.emplace_back(size - 1, "tail");
00241         }
00242         event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
00243         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00244         event.colorNodes.emplace_back(chosenNode);
00245         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00246         event.lines = {0, 1};
00247
00248         events.emplace_back(event);
00249
00250         event.reset();
00251
00252         event.titleNodes.emplace_back(chosenNode, "temp");
00253         if (size == 1)
00254             event.titleNodes.emplace_back(0, "head|tail");
00255         else if (size > 1){
00256             event.titleNodes.emplace_back(0, "head");
00257             event.titleNodes.emplace_back(size - 1, "tail");
00258         }
00259         event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
00260         event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00261         event.colorNodes.emplace_back(chosenNode);
00262         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00263         event.lines = {2};
00264
00265         events.emplace_back(event);
00266
00267         event.reset();
00268         event.titleNodes.emplace_back(chosenNode, "temp");
00269         if (size == 1)
00270             event.titleNodes.emplace_back(0, "head|tail");
00271         else if (size > 1){
00272             event.titleNodes.emplace_back(0, "head");
00273             event.titleNodes.emplace_back(size - 1, "tail");
00274         }
00275         event.colorArrows = {
00276             // {chosenNode, NodeInfo::ArrowType::LEFT},
00277             {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
00278         };
00279         event.colorNodes.emplace_back(chosenNode - 1);
00280         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00281         event.lines = {3};
00282
00283         events.emplace_back(event);
00284
00285         event.reset();
00286         event.titleNodes = {
00287             {chosenNode, "tail"},
00288             {0, "head"}
00289         };
00290         event.colorNodes = {chosenNode};
00291         event.lines = {4};
00292
00293         events.emplace_back(event);
00294     }
00295     else {
00296         this->linkedList->initHighlighter(
00297             constants::Highlighter::DLL::CODES_PATH[2].second,
00298             constants::Highlighter::DLL::CODES_PATH[2].first
00299         );
00300
00301         event.titleNodes = {
00302             {chosenNode, "temp"},
00303             {0, "head"},
00304             {size, "tail"}
00305         };
00306         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00307         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);

```

```

00308     event.colorNodes.emplace_back(chosenNode);
00309     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00310     event.lines = {0, 1};
00311
00312     events.emplace_back(event);
00313
00314     event.reset();
00315     event.titleNodes = {
00316         {chosenNode, "temp"},
00317         {0, "head|current"},
00318         {size, "tail"}
00319     };
00320     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00321     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00322     event.colorNodes.emplace_back(0);
00323     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00324     event.lines = {2};
00325
00326     events.emplace_back(event);
00327
00328     for (int i = 0; i < chosenNode; ++i) {
00329         event.reset();
00330         event.titleNodes = {
00331             {chosenNode, "temp"},
00332             {0, "head"},
00333             {size, "tail"},
00334             {i, "current"}
00335         };
00336         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00337         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00338         event.colorNodes.emplace_back(i);
00339         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00340         event.lines = {3};
00341
00342         events.emplace_back(event);
00343
00344         if (i == chosenNode - 1)
00345             break;
00346
00347         event.reset();
00348         event.titleNodes = {
00349             {chosenNode, "temp"},
00350             {0, "head"},
00351             {size, "tail"},
00352             {i, "current"}
00353         };
00354         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00355         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00356         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00357         // event.colorArrows.emplace_back(i + 1 + (i + 1 == chosenNode),
NodeInfo::ArrowType::LEFT);
00358         event.colorNodes.emplace_back(i);
00359         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00360         event.lines = {4};
00361
00362         events.emplace_back(event);
00363     }
00364
00365     event.reset();
00366     event.titleNodes = {
00367         {chosenNode, "temp"},
00368         {0, "head"},
00369         {size, "tail"},
00370         {chosenNode - 1, "current"}
00371     };
00372     event.colorArrows = {
00373         {chosenNode, NodeInfo::ArrowType::RIGHT},
00374         {chosenNode, NodeInfo::ArrowType::LEFT}
00375     };
00376     event.colorNodes.emplace_back(chosenNode);
00377     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00378     event.isPrintNormal = true;
00379     event.lines = {5, 6};
00380
00381     events.emplace_back(event);
00382
00383     event.reset();
00384     event.titleNodes = {
00385         {chosenNode, "temp"},
00386         {0, "head"},
00387         {size, "tail"}
00388     };
00389     event.colorNodes.emplace_back(chosenNode);
00390     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00391     event.lines = {7, 8};
00392
00393     events.emplace_back(event);

```

```

00394     }
00395
00396     return events;
00397 }

```

### 7.11.3.2 deleteModeEvents()

```

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

```

Definition at line 399 of file [DLLScene.cpp](#).

```

00399                                     {
00400     this->linkedList->resetEvents();
00401     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00402         return {};
00403
00404     std::vector<EventAnimation> events;
00405     EventAnimation event;
00406     int size = this->linkedList->getSize();
00407
00408     if (chosenNode == 0) {
00409         this->linkedList->initHighlighter(
00410             constants::Highlighter::DLL::CODES_PATH[3].second,
00411             constants::Highlighter::DLL::CODES_PATH[3].first
00412         );
00413
00414         if (size == 1) {
00415             event.titleNodes.emplace_back(chosenNode, "head|tail|temp");
00416             event.colorNodes.emplace_back(chosenNode);
00417             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00418             event.lines = {0, 1, 2};
00419
00420             events.emplace_back(event);
00421
00422             event.reset();
00423             event.statusChosenNode = NodeInfo::StatusNode::Visible;
00424             event.lines = {5, 6, 7};
00425
00426             events.emplace_back(event);
00427         }
00428         else {
00429             event.titleNodes = {
00430                 {chosenNode, "head|temp"},
00431                 {size - 1, "tail"}
00432             };
00433             event.colorNodes.emplace_back(chosenNode);
00434             event.lines = {0, 1};
00435
00436             events.emplace_back(event);
00437
00438             event.reset();
00439             if (size == 2)
00440                 event.titleNodes.emplace_back(size - 1, "head|tail");
00441             else
00442                 event.titleNodes = {
00443                     {size - 1, "tail"},
00444                     {chosenNode + 1, "head" }
00445                 };
00446             event.titleNodes.emplace_back(chosenNode, "temp");
00447             event.colorNodes.emplace_back(chosenNode + 1);
00448             // event.isPrintNormal = true;
00449             // event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00450             event.lines = {2};
00451
00452             events.emplace_back(event);
00453
00454             event.reset();
00455             if (size == 2)
00456                 event.titleNodes.emplace_back(size - 1, "head|tail");
00457             else
00458                 event.titleNodes = {
00459                     {size - 1, "tail"},
00460                     {chosenNode + 1, "head" }
00461                 };
00462             event.titleNodes.emplace_back(chosenNode, "temp");
00463             event.colorNodes.emplace_back(chosenNode);
00464             event.hiddenArrows = {
00465                 {chosenNode, NodeInfo::ArrowType::RIGHT},

```

```

00466         {chosenNode + 1, NodeInfo::ArrowType::LEFT}
00467     };
00468     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00469     event.isPrintNormal = true;
00470     event.lines = {3, 4};
00471
00472     events.emplace_back(event);
00473
00474     event.reset();
00475     if (size == 2)
00476         event.titleNodes.emplace_back(size - 1, "head|tail");
00477     else
00478         event.titleNodes = {
00479             {size - 1, "tail"},
00480             {chosenNode + 1, "head" }
00481         };
00482     event.hiddenArrows = {
00483         {chosenNode + 1, NodeInfo::ArrowType::LEFT}
00484     };
00485     event.statusChosenNode = NodeInfo::StatusNode::Visible;
00486     event.lines = {7};
00487     events.emplace_back(event);
00488 }
00489 }
00490
00491 else if (chosenNode == size - 1) {
00492     this->linkedList->initHighlighter(
00493         constants::Highlighter::DLL::CODES_PATH[4].second,
00494         constants::Highlighter::DLL::CODES_PATH[4].first
00495     );
00496
00497     event.titleNodes = {
00498         {0, "head"},
00499         {chosenNode, "tail|temp"}
00500     };
00501     event.colorNodes.emplace_back(chosenNode);
00502     event.lines = {0, 1};
00503
00504     events.emplace_back(event);
00505
00506     event.reset();
00507     if (size == 2)
00508         event.titleNodes.emplace_back(0, "head|tail");
00509     else
00510         event.titleNodes = {
00511             {chosenNode - 1, "tail"},
00512             {0, "head" }
00513         };
00514     event.titleNodes.emplace_back(chosenNode, "temp");
00515     event.colorNodes.emplace_back(chosenNode - 1);
00516     event.lines = {2};
00517
00518     events.emplace_back(event);
00519
00520     event.reset();
00521     if (size == 2)
00522         event.titleNodes.emplace_back(0, "head|tail");
00523     else
00524         event.titleNodes = {
00525             {chosenNode - 1, "tail"},
00526             {0, "head" }
00527         };
00528     event.titleNodes.emplace_back(chosenNode, "temp");
00529     event.colorNodes.emplace_back(chosenNode);
00530     event.hiddenArrows = {
00531         {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
00532     };
00533     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00534     event.isPrintNormal = true;
00535     event.lines = {3};
00536
00537     events.emplace_back(event);
00538
00539     event.reset();
00540     if (size == 2)
00541         event.titleNodes.emplace_back(0, "head|tail");
00542     else
00543         event.titleNodes = {
00544             {chosenNode - 1, "tail"},
00545             {0, "head" }
00546         };
00547     event.hiddenArrows = {
00548         {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
00549     };
00550     event.statusChosenNode = NodeInfo::StatusNode::Visible;
00551     event.lines = {4};
00552

```

```

00553         events.emplace_back(event);
00554     }
00555     else {
00556         this->linkedList->initHighlighter(
00557             constants::Highlighter::DLL::CODES_PATH[5].second,
00558             constants::Highlighter::DLL::CODES_PATH[5].first
00559         );
00560
00561         event.titleNodes = {
00562             {0, "head|temp"},
00563             {size - 1, "tail"}
00564         };
00565         event.colorNodes.emplace_back(0);
00566         event.lines = {0, 1};
00567
00568         events.emplace_back(event);
00569
00570         for (int i = 0; i <= chosenNode; ++i) {
00571             event.reset();
00572             event.titleNodes = {
00573                 {0, "head"},
00574                 {i, "temp"},
00575                 {size - 1, "tail"}
00576             };
00577             event.colorNodes.emplace_back(i);
00578             event.lines = {2};
00579
00580             events.emplace_back(event);
00581
00582             if (i == chosenNode)
00583                 break;
00584
00585             event.reset();
00586             event.titleNodes = {
00587                 {0, "head"},
00588                 {i, "temp"},
00589                 {size - 1, "tail"}
00590             };
00591             event.colorNodes.emplace_back(i);
00592             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00593             event.lines = {3};
00594
00595             events.emplace_back(event);
00596         }
00597
00598         event.reset();
00599         event.titleNodes = {
00600             {0, "head"},
00601             {chosenNode, "temp"},
00602             {size - 1, "tail"}
00603         };
00604         event.colorNodes.emplace_back(chosenNode);
00605         event.colorArrows = {
00606             {chosenNode - 1, NodeInfo::ArrowType::RIGHT},
00607             {chosenNode + 1, NodeInfo::ArrowType::LEFT}
00608         };
00609         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00610         event.isPrintNormal = true;
00611         event.lines = {4, 5};
00612
00613         events.emplace_back(event);
00614
00615         event.reset();
00616         event.titleNodes = {
00617             {0, "head"},
00618             {size - 1, "tail"}
00619         };
00620         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00621         event.lines = {6};
00622
00623         events.emplace_back(event);
00624     }
00625
00626     return events;
00627 }

```

### 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](#).

```
00011                                     {
00012     if (this->isMenuOpen)
00013         this->menu->pollEvents(event, mousePosView);
00014
00015     this->controlMenu->pollEvents(event, mousePosView);
00016 }
```

#### 7.11.3.4 render()

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

Implements [BaseScene](#).

Definition at line 123 of file [DLLScene.cpp](#).

```
00123     {
00124     if (this->isMenuOpen)
00125         this->menu->render();
00126
00127     if (this->isDemoCodeOpen)
00128         this->linkedList->renderHighlighter();
00129
00130     this->controlMenu->render();
00131     this->linkedList->render();
00132 }
```

#### 7.11.3.5 reset()

```
void DLLScene::reset ( )
```

Definition at line 139 of file [DLLScene.cpp](#).

```
00139     {
00140     this->menu->resetActiveOptionsMenu();
00141 }
```

#### 7.11.3.6 searchModeEvents()

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

Definition at line 717 of file [DLLScene.cpp](#).

```
00717                                     {
00718     this->linkedList->resetEvents();
00719     this->linkedList->initHighlighter(
00720         constants::Highlighter::DLL::CODES_PATH[7].second,
00721         constants::Highlighter::DLL::CODES_PATH[7].first
00722     );
00723
00724     std::vector<EventAnimation> events;
00725     EventAnimation event;
00726     int size = this->linkedList->getSize();
00727
00728     if (size > 1)
00729         event.titleNodes = {
00730             {0, "head|current"},
00731             {size - 1, "tail"}
00732         };
00733 }
```

```

00733     else
00734         event.titleNodes = {
00735             {0, "head|tail|current"}
00736         };
00737     event.colorNodes.push_back(0);
00738     event.lines = {0};
00739
00740     events.emplace_back(event);
00741
00742     for (int i = 0; i <= chosenNode; ++i) {
00743         if (i == chosenNode && chosenNode == this->linkedList->getSize())
00744             break;
00745
00746         event.reset();
00747         event.titleNodes = {
00748             {0, "head"},
00749             {size - 1, "tail"},
00750             {i, "current"}
00751         };
00752         event.colorNodes.push_back(i);
00753         event.lines = {1};
00754
00755         events.emplace_back(event);
00756
00757         if (i == chosenNode) break;
00758
00759         event.reset();
00760         event.titleNodes = {
00761             {0, "head"},
00762             {size - 1, "tail"},
00763             {i, "current"}
00764         };
00765         event.colorNodes.push_back(i);
00766         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00767         event.lines = {4};
00768
00769         events.emplace_back(event);
00770     }
00771
00772     if (chosenNode == this->linkedList->getSize()) {
00773         event.reset();
00774         event.titleNodes.emplace_back(0, "head");
00775         event.titleNodes.emplace_back(size - 1, "tail");
00776         event.lines = {5};
00777
00778         events.emplace_back(event);
00779     } else {
00780         event.reset();
00781         event.titleNodes = {
00782             {0, "head"},
00783             {size - 1, "tail"},
00784             {chosenNode, "current"}
00785         };
00786         event.colorNodes.push_back(chosenNode);
00787         event.lines = {2, 3};
00788
00789         events.emplace_back(event);
00790     }
00791
00792     return events;
00793 }

```

### 7.11.3.7 update()

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

Implements [BaseScene](#).

Definition at line 18 of file [DLLScene.cpp](#).

```

00018     {
00019         if (this->isMenuOpen) {
00020             this->menu->update();
00021
00022             constants::MenuLinkedList::Button status = this->menu->getActiveOptionsMenu();
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);
00051             std::string token;
00052             while (std::getline(ss, token, ','))
00053                 values.push_back(token);
00054             this->linkedList->createLinkedList(values);
00055             this->menu->createModeValue[2] = "None";
00056         }
00057         this->controlMenu->reset();
00058         break;
00059     case constants::MenuLinkedList::Button::ADD_BUTTON:
00060         if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
this->menu->addModeValue[0].empty())
00061             break;
00062
00063         this->linkedList->addNode(
00064             std::stoi(this->menu->addModeValue[0]),
00065             this->menu->addModeValue[1],
00066             this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00067         );
00068
00069         std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
<< std::endl;
00070         this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00071         this->controlMenu->reset();
00072         break;
00073     case constants::MenuLinkedList::Button::DELETE_BUTTON:
00074         if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
00075             break;
00076
00077         this->linkedList->deleteNode(
00078             std::stoi(this->menu->deleteModeValue),
00079             this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
00080         );
00081
00082         std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00083         this->menu->deleteModeValue = "None";
00084         this->controlMenu->reset();
00085         break;
00086     case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00087         if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
00088             break;
00089
00090         this->linkedList->updateNode(
00091             std::stoi(this->menu->updateModeValue[0]),
00092             this->menu->updateModeValue[1],
00093             this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00094         );
00095
00096         std::cout << "Update: " << this->menu->updateModeValue[0] << " " <<
this->menu->updateModeValue[1] << std::endl;
00097         this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00098         this->controlMenu->reset();
00099         break;
00100     case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00101         if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00102             break;
00103
00104         this->linkedList->searchNode(
00105             this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
00106         );
00107

```

```

00108         std::cout << "Search: " << this->menu->searchModeValue << std::endl;
00109         this->menu->searchModeValue = "None";
00110         this->controlMenu->reset();
00111         break;
00112     }
00113 }
00114
00115 this->controlMenu->update();
00116
00117 this->linkedList->processControlMenu(this->controlMenu->getStatus());
00118 this->linkedList->setSpeed(this->controlMenu->getSpeed());
00119
00120 this->linkedList->update();
00121 }

```

### 7.11.3.8 updateModeEvents()

```

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

```

Definition at line 629 of file [DLLScene.cpp](#).

```

00629                                     {
00630     this->linkedList->resetEvents();
00631     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00632         return {};
00633
00634     this->linkedList->initHighlighter(
00635         constants::Highlighter::DLL::CODES_PATH[6].second,
00636         constants::Highlighter::DLL::CODES_PATH[6].first
00637     );
00638
00639     std::vector<EventAnimation> events;
00640     EventAnimation event;
00641     int size = this->linkedList->getSize();
00642
00643     if (size > 1)
00644         event.titleNodes = {
00645             {0, "head|current"},
00646             {size - 1, "tail"}
00647         };
00648     else
00649         event.titleNodes = {
00650             {0, "head|tail|current"}
00651         };
00652     event.colorNodes.push_back(0);
00653     event.isPrintPreVal = true;
00654     event.lines = {0};
00655
00656     events.emplace_back(event);
00657
00658     if (chosenNode) {
00659         for (int i = 0; i <= chosenNode; ++i) {
00660             event.reset();
00661             event.titleNodes = {
00662                 {0, "head"},
00663                 {size - 1, "tail"},
00664                 {i, "current"},
00665             };
00666             event.colorNodes.push_back(i);
00667             event.isPrintPreVal = true;
00668             event.lines = {1};
00669
00670             events.emplace_back(event);
00671
00672             if (i == chosenNode) break;
00673
00674             event.reset();
00675             event.titleNodes = {
00676                 {0, "head"},
00677                 {i, "current"},
00678                 {size - 1, "tail"}
00679             };
00680             event.colorNodes.push_back(i);
00681             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00682             event.isPrintPreVal = true;
00683             event.lines = {2};
00684
00685             events.emplace_back(event);

```

```

00686     }
00687 }
00688
00689 event.reset();
00690 if (size == 1)
00691     event.titleNodes = {
00692         {0, "head|tail|current"}
00693     };
00694 else if (chosenNode == size - 1)
00695     event.titleNodes = {
00696         {0, "head"},
00697         {chosenNode, "current|tail"}
00698     };
00699 else if (chosenNode == 0)
00700     event.titleNodes = {
00701         {0, "head|current"},
00702         {size - 1, "tail"}
00703     };
00704 else
00705     event.titleNodes = {
00706         {0, "head"},
00707         {chosenNode, "current"},
00708         {size - 1, "tail"}
00709     };
00710 event.lines = {3};
00711 events.emplace_back(event);
00712
00713 return events;
00714 }
00715 }

```

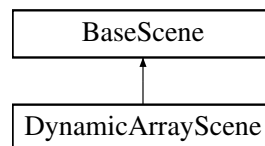
The documentation for this class was generated from the following files:

- [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](#).

```
00007                                     : BaseScene(window) {
00008     this->init();
00009 }
```

### 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](#).

```
00156
00157     this->array->resetEvents();
00158     if (chosenNode < 0 || chosenNode > this->array->getSize())
00159         return {};
00160
00161     // init highlighter
00162     // ...
00163
00164     int size = this->array->getSize() + 1,
00165         squaresSize = this->array->getSquaresSize();
00166     std::vector<EventAnimation> events;
00167     EventAnimation event;
00168
00169     if (size > squaresSize) {
00170         ++squaresSize;
00171         event.eventSquares.assign(squaresSize, EventSquare());
00172         event.eventSquaresTemp.assign(squaresSize, EventSquare());
00173         for (auto &square : event.eventSquares) {
00174             square.status = Square::Status::active;
00175             square.isPrintPreVal = true;
00176         }
00177         event.eventSquares.back().status = Square::Status::hidden;
00178         if (size > 1)
00179             event.eventSquares[size - 2].title = "n";
00180         for (auto &square : event.eventSquaresTemp) {
00181             square.status = Square::Status::inactive;
00182             square.isPrintPreVal = true;
00183         }
00184
00185         events.emplace_back(event);
00186
00187         for (int i = 0; i < size - 1; ++i) {
00188             event = EventAnimation();
00189             event.eventSquares.assign(squaresSize, EventSquare());
00190             event.eventSquaresTemp.assign(squaresSize, EventSquare());
00191             for (auto &square : event.eventSquares) {
00192                 square.status = Square::Status::active;
00193                 square.isPrintPreVal = true;
00194             }
00195             event.eventSquares.back().status = Square::Status::hidden;
00196             if (size > 1)
00197                 event.eventSquares[size - 2].title = "n";
00198             for (auto &square : event.eventSquaresTemp) {
00199                 square.status = Square::Status::inactive;
00200                 square.isPrintPreVal = true;
00201             }
00202             for (int j = 0; j < i; ++j) {
00203                 event.eventSquaresTemp[j].status = Square::Status::active;
00204                 event.eventSquaresTemp[j].isPrintPreVal = false;
00205             }
00206             event.eventSquaresTemp[i].status = Square::Status::chosen;
00207             event.eventSquaresTemp[i].title = "m";
00208
00209             events.emplace_back(event);
00210
00211             event.eventSquaresTemp[i].isPrintPreVal = false;
00212             event.eventSquares[i].status = Square::Status::chosen;
00213
00214             events.emplace_back(event);
00215         }
00216     }
00217
00218     event = EventAnimation();
00219     event.eventSquares.assign(squaresSize, EventSquare());
00220     event.eventSquaresTemp.assign(squaresSize, EventSquare());
00221     for (auto &square : event.eventSquares) {
00222         square.status = Square::Status::active;
00223         square.isPrintPreVal = true;
00224     }
00225     for (int i = size - 1; i < squaresSize; ++i)
00226         event.eventSquares[i].status = Square::Status::inactive;
00227     if (size > 1)
00228         event.eventSquares[size - 2].title = "n";
00229     for (auto &square : event.eventSquaresTemp) {
00230         square.status = Square::Status::hidden;
00231     }
00232
00233     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     }

```

```

00287     int size = this->array->getSize(),
00288         squaresSize = this->array->getSquaresSize();
00289     std::vector<EventAnimation> events;
00290     EventAnimation event;
00291
00292     for (int i = chosenNode; i < size - 1; ++i) {
00293         event = EventAnimation();
00294         event.eventSquares.assign(squaresSize, EventSquare());
00295         for (auto &square : event.eventSquares) {
00296             square.status = Square::Status::active;
00297             square.isPrintPreVal = true;
00298         }
00299         for (int j = size; j < squaresSize; ++j)
00300             event.eventSquares[j].status = Square::Status::inactive;
00301         for (int j = 0; j < i; ++j)
00302             event.eventSquares[j].isPrintPreVal = false;
00303         event.eventSquares[i].status = Square::Status::chosen;
00304         for (auto &square : event.eventSquaresTemp)
00305             square.status = Square::Status::hidden;
00306         event.eventSquares[size - 1].title = "n";
00307
00308         events.emplace_back(event);
00309
00310         event.eventSquares[i].isPrintPreVal = false;
00311         event.eventSquares[i + 1].status = Square::Status::chosen;
00312
00313         events.emplace_back(event);
00314     }
00315
00316     event = EventAnimation();
00317     event.eventSquares.assign(squaresSize, EventSquare());
00318     for (int i = 0; i < size - 1; ++i) {
00319         event.eventSquares[i].status = Square::Status::active;
00320         if (i == size - 2)
00321             event.eventSquares[i].title = "n";
00322     }
00323     for (int i = size - 1; i < squaresSize; ++i)
00324         event.eventSquares[i].status = Square::Status::inactive;
00325
00326     events.emplace_back(event);
00327
00328     return events;
00329 }

```

#### 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](#).

```

00140
00141     if (this->isMenuOpen)
00142         this->menu->pollEvents(event, mousePosView);
00143
00144     this->controlMenu->pollEvents(event, mousePosView);
00145 }

```

#### 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             if (this->isDemoCodeOpen)
00133                 this->array->renderHighlighter();
00134             this->controlMenu->render();
00135             this->array->render();
00136         }
00137     }
00138 }

```

### 7.12.3.6 reset()

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

Definition at line 152 of file [DynamicArrayScene.cpp](#).

```

00152         {
00153             this->menu->resetActiveOptionsMenu();
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             // init highlighter
00364             // ...
00365             int size = this->array->getSize(),
00366                 squaresSize = this->array->getSquaresSize();
00367             std::vector<EventAnimation> events;
00368             EventAnimation event;
00369             for (int i = 0; i <= chosenNode; ++i) {
00370                 if (i == size) break;
00371                 event = EventAnimation();
00372                 event.eventSquares.assign(squaresSize, EventSquare());
00373                 for (int j = 0; j < size; ++j) {
00374                     event.eventSquares[j].status = Square::Status::active;
00375                     if (j == size - 1)
00376                         event.eventSquares[size - 1].title = "n";
00377                 }
00378                 event.eventSquares[i].status = Square::Status::chosen;
00379                 events.emplace_back(event);
00380             }
00381             if (chosenNode == size) {
00382                 event = EventAnimation();
00383                 event.eventSquares.assign(squaresSize, EventSquare());
00384                 for (int j = 0; j < size; ++j) {
00385                     event.eventSquares[j].status = Square::Status::active;
00386                     if (j == size - 1)
00387                         event.eventSquares[size - 1].title = "n";
00388                 }
00389                 events.emplace_back(event);
00390             }
00391             return events;
00392         }
00393     }
00394 }

```

### 7.12.3.8 update()

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

Implements [BaseScene](#).

Definition at line 11 of file [DynamicArrayScene.cpp](#).

```
00011         {
00012             if (this->isMenuOpen) {
00013                 this->menu->update();
00014
00015                 constants::MenuArray::Button status = this->menu->getActiveOptionsMenu();
00016                 constants::MenuArray::CreateMode::Button createMode;
00017                 switch (status){
00018                     case constants::MenuArray::Button::CREATE_BUTTON:
00019                         createMode = this->menu->getActiveCreateMode();
00020                         if (createMode == constants::MenuArray::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->array->createArray(size);
00027                         } else if (createMode ==
00028 constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
00029                             if (this->menu->createModeValue[1] == "None")
00030                                 break;
00031                             std::vector<std::string> values;
00032                             std::string value = this->menu->createModeValue[1];
00033                             std::stringstream ss(value);
00034                             std::string token;
00035                             while (std::getline(ss, token, ',')) {
00036                                 values.push_back(token);
00037                             }
00038                             this->array->createArray(values);
00039                         } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
00040                             if (this->menu->createModeValue[2] == "None")
00041                                 break;
00042                             std::vector<std::string> values;
00043                             std::string value = this->menu->createModeValue[2];
00044                             std::stringstream ss(value);
00045                             std::string token;
00046                             while (std::getline(ss, token, ','))
00047                                 values.push_back(token);
00048                             this->array->createArray(values);
00049                             this->menu->createModeValue[2] = "None";
00050                         }
00051                         this->controlMenu->reset();
00052                         break;
00053                     case constants::MenuArray::Button::ADD_BUTTON:
00054                         if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
00055 this->menu->addModeValue[0].empty())
00056                             break;
00057                         this->array->addSquare(
00058                             std::stoi(this->menu->addModeValue[0]),
00059                             this->menu->addModeValue[1],
00060                             this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00061                         );
00062                         std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
00063 << std::endl;
00064                         this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00065                         this->controlMenu->reset();
00066                         break;
00067                     case constants::MenuArray::Button::DELETE_BUTTON:
00068                         if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
00069                             break;
00070                         this->array->deleteSquare(
00071                             std::stoi(this->menu->deleteModeValue),
00072                             this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
00073                         );
00074                         std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00075                         this->menu->deleteModeValue = "None";
00076                         this->controlMenu->reset();
00077                         break;
00078                     case constants::MenuArray::Button::UPDATE_BUTTON:
00079                         if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
00080 "None" || this->menu->updateModeValue[0].empty())
00081                             break;
00082
```

```

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         }

```

```

00356     events.emplace_back(event);
00357
00358     return events;
00359 }

```

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](#).

```
00810     {  
00811         stop();  
00812     }
```

### 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](#).

```
00699     {  
00700     #if __WIN32  
00701         if (m_future.valid())  
00702         {  
00703             // Close all windows that werent open when we started the future  
00704             auto previous_windows = m_windows;  
00705             EnumWindows(&enum_windows_callback, (LPARAM)this);  
00706             for (auto hwnd : m_windows)  
00707                 if (previous_windows.find(hwnd) == previous_windows.end())  
00708                 {  
00709                     SendMessage(hwnd, WM_CLOSE, 0, 0);  
00710                     // Also send IDNO in case of a Yes/No or Abort/Retry/Ignore messagebox  
00711                     SendMessage(hwnd, WM_COMMAND, IDNO, 0);  
00712                 }  
00713         }  
00714     #elif __EMSCRIPTEN__ || __NX__  
00715         // FIXME: do something  
00716         return false; // cannot kill  
00717     #else  
00718         ::kill(m_pid, SIGKILL);  
00719     #endif  
00720         stop();  
00721         return true;  
00722     }
```

## 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](#).

```
00815     {
00816         if (!m_running)
00817             return true;
00818
00819 #if _WIN32
00820         if (m_future.valid())
00821         {
00822             auto status = m_future.wait_for(std::chrono::milliseconds(timeout));
00823             if (status != std::future_status::ready)
00824             {
00825                 // On Windows, we need to run the message pump. If the async
00826                 // thread uses a Windows API dialog, it may be attached to the
00827                 // main thread and waiting for messages that only we can dispatch.
00828                 MSG msg;
00829                 while (PeekMessage(&msg, nullptr, 0, 0, PM_REMOVE))
00830                 {
00831                     TranslateMessage(&msg);
00832                     DispatchMessage(&msg);
00833                 }
00834                 return false;
00835             }
00836
00837             m_stdout = m_future.get();
00838         }
00839 #elif __EMSCRIPTEN__ || __NX__
00840         // FIXME: do something
00841         (void)timeout;
00842 #else
00843         char buf[BUFSIZ];
00844         ssize_t received = read(m_fd, buf, BUFSIZ); // Flawfinder: ignore
00845         if (received > 0)
00846         {
00847             m_stdout += std::string(buf, received);
00848             return false;
00849         }
00850
00851         // Reap child process if it is dead. It is possible that the system has already reaped it
00852         // (this happens when the calling application handles or ignores SIG_CHLD) and results in
00853         // waitpid() failing with ECHILD. Otherwise we assume the child is running and we sleep for
00854         // a little while.
00855         int status;
00856         pid_t child = waitpid(m_pid, &status, WNOHANG);
00857         if (child != m_pid && (child >= 0 || errno != ECHILD))
00858         {
00859             // FIXME: this happens almost always at first iteration
00860             std::this_thread::sleep_for(std::chrono::milliseconds(timeout));
00861             return false;
00862         }
00863
00864         close(m_fd);
00865         m_exit_code = WEXITSTATUS(status);
00866 #endif
00867
00868         m_running = false;
00869         return true;
00870     }
```

## 7.15.3.3 result()

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

Definition at line 690 of file [FileDialog.h](#).

```
00691     {
00692         stop();
00693         if (exit_code)
00694             *exit_code = m_exit_code;
00695         return m_stdout;
00696     }
```

### 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](#).

```
00765     {
00766         stop();
00767         m_stdout.clear();
00768         m_exit_code = -1;
00769
00770         int in[2], out[2];
00771         if (pipe(in) != 0 || pipe(out) != 0)
00772             return;
00773
00774         m_pid = fork();
00775         if (m_pid < 0)
00776             return;
00777
00778         close(in[m_pid ? 0 : 1]);
00779         close(out[m_pid ? 1 : 0]);
00780
00781         if (m_pid == 0)
00782         {
00783             dup2(in[0], STDIN_FILENO);
00784             dup2(out[1], STDOUT_FILENO);
00785
00786             // Ignore stderr so that it doesnt pollute the console (e.g. GTK+ errors from zenity)
00787             int fd = open("/dev/null", O_WRONLY);
00788             dup2(fd, STDERR_FILENO);
00789             close(fd);
00790
00791             std::vector<char *> args;
00792             std::transform(command.cbegin(), command.cend(), std::back_inserter(args),
00793                 [](std::string const &s) { return const_cast<char *>(s.c_str()); });
00794             args.push_back(nullptr); // null-terminate argv[]
00795
00796             execvp(args[0], args.data());
00797             exit(1);
00798         }
00799
00800         close(in[1]);
00801         m_fd = out[0];
00802         auto flags = fcntl(m_fd, F_GETFL);
00803         fcntl(m_fd, F_SETFL, flags | O_NONBLOCK);
00804
00805         m_running = true;
00806     }
```

### 7.15.3.5 stop()

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

Definition at line 872 of file [FileDialog.h](#).

```
00873     {
00874         // Loop until the user closes the dialog
00875         while (!ready())
00876             ;
00877     }
```

## 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](#).

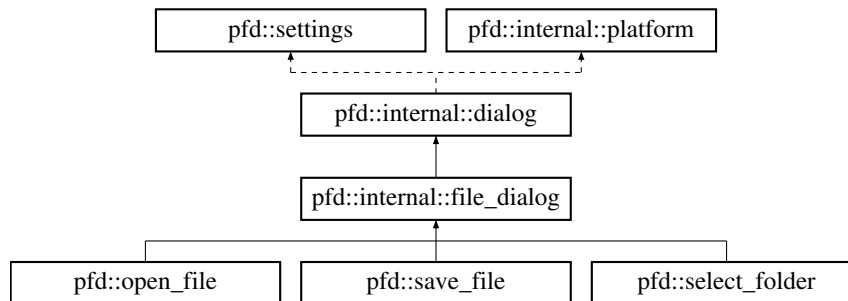
The documentation for this class was generated from the following file:

- [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](#).

```
00290     {
00291         open,
00292         save,
00293         folder,
00294     };
```

## 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](#).

```
01063     {
01064     #if _WIN32
01065         std::string filter_list;
01066         std::regex whitespace(" *");
01067         for (size_t i = 0; i + 1 < filters.size(); i += 2)
01068         {
01069             filter_list += filters[i] + '\\0';
01070             filter_list += std::regex_replace(filters[i + 1], whitespace, ";") + '\\0';
01071         }
01072         filter_list += '\\0';
01073
01074         m_async->start_func([this, in_type, title, default_path, filter_list,
01075                             options](int *exit_code) -> std::string
01076         {
01077             (void)exit_code;
01078             m_wtitle = internal::str2wstr(title);
01079             m_wdefault_path = internal::str2wstr(default_path);
01080             auto wfilter_list = internal::str2wstr(filter_list);
01081
01082             // Initialise COM. This is required for the new folder selection window,
01083             // (see https://github.com/samhocevar/portable-file-dialogs/pull/21)
01084             // and to avoid random crashes with GetOpenFileNameW() (see
01085             // https://github.com/samhocevar/portable-file-dialogs/issues/51)
01086             ole32_dll ole32;
01087
01088             // Folder selection uses a different method
01089             if (in_type == type::folder)
01090             {
01091             #if PFD_HAS_IFILEDIALOG
01092                 if (flags(flag::is_vista))
01093                 {
01094                     // On Vista and higher we should be able to use IFileDialog for folder selection
01095                     IFileDialog *ifd;
01096                     HRESULT hr = dll::proc<HRESULT WINAPI (REFCLSID, LPUNKNOWN, DWORD, REFIID, LPVOID
01097 *)>(ole32, "CoCreateInstance")
01098                                     (CLSID_FileOpenDialog, nullptr, CLSCTX_INPROC_SERVER,
01099                                     IID_PPV_ARGS(&ifd));
01098
01099                     // In case CoCreateInstance fails (which it should not), try legacy approach
01100                     if (SUCCEEDED(hr))
01101                         return select_folder_vista(ifd, options & opt::force_path);
01102                 }
01102             }
```

```

01103 #endif
01104
01105     BROWSEINFOW bi;
01106     memset(&bi, 0, sizeof(bi));
01107
01108     bi.lpfn = &bffcallback;
01109     bi.lParam = (LPARAM)this;
01110
01111     if (flags(flag::is_vista))
01112     {
01113         if (ole32.is_initialized())
01114             bi.ulFlags |= BIF_NEWDIALOGSTYLE;
01115         bi.ulFlags |= BIF_EDITBOX;
01116         bi.ulFlags |= BIF_STATUSTEXT;
01117     }
01118
01119     auto *list = SHBrowseForFolderW(&bi);
01120     std::string ret;
01121     if (list)
01122     {
01123         auto buffer = new wchar_t[MAX_PATH];
01124         SHGetPathFromIDListW(list, buffer);
01125         dll::proc<void WINAPI (LPVOID)>(ole32, "CoTaskMemFree")(list);
01126         ret = internal::wstr2str(buffer);
01127         delete[] buffer;
01128     }
01129     return ret;
01130 }
01131
01132 OPENFILENAMEW ofn;
01133 memset(&ofn, 0, sizeof(ofn));
01134 ofn.lStructSize = sizeof(OPENFILENAMEW);
01135 ofn.hwndOwner = GetActiveWindow();
01136
01137 ofn.lpstrFilter = wfilter_list.c_str();
01138
01139 auto woutput = std::wstring(MAX_PATH * 256, L'\0');
01140 ofn.lpstrFile = (LPWSTR)woutput.data();
01141 ofn.nMaxFile = (DWORD)woutput.size();
01142 if (!m_wdefault_path.empty())
01143 {
01144     // If a directory was provided, use it as the initial directory. If
01145     // a valid path was provided, use it as the initial file. Otherwise,
01146     // let the Windows API decide.
01147     auto path_attr = GetFileAttributesW(m_wdefault_path.c_str());
01148     if (path_attr != INVALID_FILE_ATTRIBUTES && (path_attr & FILE_ATTRIBUTE_DIRECTORY))
01149         ofn.lpstrInitialDir = m_wdefault_path.c_str();
01150     else if (m_wdefault_path.size() <= woutput.size())
01151         //second argument is size of buffer, not length of string
01152         StringCchCopyW(ofn.lpstrFile, MAX_PATH*256+1, m_wdefault_path.c_str());
01153     else
01154     {
01155         ofn.lpstrFileTitle = (LPWSTR)m_wdefault_path.data();
01156         ofn.nMaxFileTitle = (DWORD)m_wdefault_path.size();
01157     }
01158 }
01159 ofn.lpstrTitle = m_wtitle.c_str();
01160 ofn.Flags = OFN_NOCHANGEDIR | OFN_EXPLORER;
01161
01162 dll comdlg32("comdlg32.dll");
01163
01164 // Apply new visual style (required for windows XP)
01165 new_style_context ctx;
01166
01167 if (in_type == type::save)
01168 {
01169     if (!(options & opt::force_overwrite))
01170         ofn.Flags |= OFN_OVERWRITEPROMPT;
01171
01172     dll::proc<BOOL WINAPI (LPOPENFILENAMEW)> get_save_file_name(comdlg32, "GetSaveFileNameW");
01173     if (get_save_file_name(&ofn) == 0)
01174         return "";
01175     return internal::wstr2str(woutput.c_str());
01176 }
01177 else
01178 {
01179     if (options & opt::multiselect)
01180         ofn.Flags |= OFN_ALLOWMULTISELECT;
01181     ofn.Flags |= OFN_PATHMUSTEXIST;
01182
01183     dll::proc<BOOL WINAPI (LPOPENFILENAMEW)> get_open_file_name(comdlg32, "GetOpenFileNameW");
01184     if (get_open_file_name(&ofn) == 0)
01185         return "";
01186 }
01187
01188 std::string prefix;
01189 for (wchar_t const *p = woutput.c_str(); *p; )

```



```

01190     {
01191         auto filename = internal::wstr2str(p);
01192         p += wcslen(p);
01193         // In multiselect mode, we advance p one wchar further and
01194         // check for another filename. If there is one and the
01195         // prefix is empty, it means we just read the prefix.
01196         if ((options & opt::multiselect) && ++p && prefix.empty())
01197         {
01198             prefix = filename + "/";
01199             continue;
01200         }
01201         m_vector_result.push_back(prefix + filename);
01202     }
01203 }
01204
01205     return "";
01206 });
01207 #elif __EMSCRIPTEN__
01208     // FIXME: do something
01209     (void)in_type;
01210     (void)title;
01211     (void)default_path;
01212     (void)filters;
01213     (void)options;
01214 #else
01215     auto command = desktop_helper();
01216
01217     if (is_osascript())
01218     {
01219         std::string script = "set ret to choose";
01220         switch (in_type)
01221         {
01222             case type::save:
01223                 script += " file name";
01224                 break;
01225             case type::open: default:
01226                 script += " file";
01227                 if (options & opt::multiselect)
01228                     script += " with multiple selections allowed";
01229                 break;
01230             case type::folder:
01231                 script += " folder";
01232                 break;
01233         }
01234
01235         if (default_path.size())
01236         {
01237             if (in_type == type::folder || is_directory(default_path))
01238                 script += " default location ";
01239             else
01240                 script += " default name ";
01241             script += osascript_quote(default_path);
01242         }
01243
01244         script += " with prompt " + osascript_quote(title);
01245
01246         if (in_type == type::open)
01247         {
01248             // Concatenate all user-provided filter patterns
01249             std::string patterns;
01250             for (size_t i = 0; i < filters.size() / 2; ++i)
01251                 patterns += " " + filters[2 * i + 1];
01252
01253             // Split the pattern list to check whether "*" is in there; if it
01254             // is, we have to disable filters because there is no mechanism in
01255             // OS X for the user to override the filter.
01256             std::regex sep("\\s+");
01257             std::string filter_list;
01258             bool has_filter = true;
01259             std::sregex_token_iterator iter(patterns.begin(), patterns.end(), sep, -1);
01260             std::sregex_token_iterator end;
01261             for (; iter != end; ++iter)
01262             {
01263                 auto pat = iter->str();
01264                 if (pat == "*" || pat == ".*")
01265                     has_filter = false;
01266                 else if (internal::starts_with(pat, ".*"))
01267                     filter_list += "," + osascript_quote(pat.substr(2, pat.size() - 2));
01268             }
01269
01270             if (has_filter && filter_list.size() > 0)
01271             {
01272                 // There is a weird AppleScript bug where file extensions of length != 3 are
01273                 // ignored, e.g. type{"txt"} works, but type{"json"} does not. Fortunately if
01274                 // the whole list starts with a 3-character extension, everything works again.
01275                 // We use "///" for such an extension because we are sure it cannot appear in
01276                 // an actual filename.

```

```

01277         script += " of type {\\"//\\" + filter_list + "}";
01278     }
01279 }
01280
01281 if (in_type == type::open && (options & opt::multiselect))
01282 {
01283     script += "\\nset s to \\";
01284     script += "\\nrepeat with i in ret";
01285     script += "\\n set s to s & (POSIX path of i) & \\";
01286     script += "\\nend repeat";
01287     script += "\\ncopy s to stdout";
01288 }
01289 else
01290 {
01291     script += "\\nPOSIX path of ret";
01292 }
01293
01294 command.push_back("-e");
01295 command.push_back(script);
01296 }
01297 else if (is_zenity())
01298 {
01299     command.push_back("--file-selection");
01300
01301     // If the default path is a directory, make sure it ends with "/" otherwise zenity will
01302     // open the file dialog in the parent directory.
01303     auto filename_arg = "--filename=" + default_path;
01304     if (in_type != type::folder && !ends_with(default_path, "/") &&
internal::is_directory(default_path))
01305         filename_arg += "/";
01306     command.push_back(filename_arg);
01307
01308     command.push_back("--title");
01309     command.push_back(title);
01310     command.push_back("--separator=\\n");
01311
01312     for (size_t i = 0; i < filters.size() / 2; ++i)
01313     {
01314         command.push_back("--file-filter");
01315         command.push_back(filters[2 * i] + "|" + filters[2 * i + 1]);
01316     }
01317
01318     if (in_type == type::save)
01319         command.push_back("--save");
01320     if (in_type == type::folder)
01321         command.push_back("--directory");
01322     if (!(options & opt::force_overwrite))
01323         command.push_back("--confirm-overwrite");
01324     if (options & opt::multiselect)
01325         command.push_back("--multiple");
01326 }
01327 else if (is_kdialog())
01328 {
01329     switch (in_type)
01330     {
01331     case type::save: command.push_back("--getsavefilename"); break;
01332     case type::open: command.push_back("--getopenfilename"); break;
01333     case type::folder: command.push_back("--getexistingdirectory"); break;
01334     }
01335     if (options & opt::multiselect)
01336     {
01337         command.push_back("--multiple");
01338         command.push_back("--separate-output");
01339     }
01340
01341     command.push_back(default_path);
01342
01343     std::string filter;
01344     for (size_t i = 0; i < filters.size() / 2; ++i)
01345         filter += (i == 0 ? "" : " | ") + filters[2 * i] + "(" + filters[2 * i + 1] + ")";
01346     command.push_back(filter);
01347
01348     command.push_back("--title");
01349     command.push_back(title);
01350 }
01351
01352 if (flags(flag::is_verbose))
01353     std::cerr << "pfd: " << command << std::endl;
01354
01355 m_async->start_process(command);
01356 #endif
01357 }

```

## 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](#).

```
01360 {
01361 #if _WIN32
01362     return m_async->result();
01363 #else
01364     auto ret = m_async->result();
01365     // Strip potential trailing newline (zenity). Also strip trailing slash
01366     // added by osascript for consistency with other backends.
01367     while (!ret.empty() && (ret.back() == '\n' || ret.back() == '/'))
01368         ret.pop_back();
01369     return ret;
01370 #endif
01371 }
```

### 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](#).

```
01374 {
01375 #if _WIN32
01376     m_async->result();
01377     return m_vector_result;
01378 #else
01379     std::vector<std::string> ret;
01380     auto result = m_async->result();
01381     for (;;)
01382     {
01383         // Split result along newline characters
01384         auto i = result.find('\n');
01385         if (i == 0 || i == std::string::npos)
01386             break;
01387         ret.push_back(result.substr(0, i));
01388         result = result.substr(i + 1, result.size());
01389     }
01390     return ret;
01391 #endif
01392 }
```

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](#).

```
00007                                     {
00008     this->window = window;
00009     this->linesCount = linesCount;
00010
00011     this->codeTexture.loadFromFile(codePath);
00012     this->codeSprite.setTexture(this->codeTexture);
00013     this->codeSprite.setScale(constants::Highlighter::codeScale);
00014
00015     this->codeSprite.setOrigin(
00016         this->codeSprite.getLocalBounds().width,
00017         this->codeSprite.getLocalBounds().height
00018     );
00019
00020     this->codeSprite.setPosition(constants::Highlighter::codePos);
00021
00022     float heightTop = 43;
00023
00024     this->rectSize = sf::Vector2f(
00025         this->codeSprite.getGlobalBounds().width,
00026         ((this->codeSprite.getLocalBounds().height - heightTop * 2) /
00027             static_cast<float>(this->linesCount)) * constants::Highlighter::codeScale.y
00028     );
00029
00030     for (int i = 0; i < this->linesCount; ++i) {
00031         sf::RectangleShape rect(this->rectSize);
00032         rect.setOrigin(rect.getLocalBounds().width, rect.getLocalBounds().height);
00033         rect.setFillColor(constants::transparentGreen);
00034         rect.setPosition(
00035             this->codeSprite.getPosition().x,
00036             this->codeSprite.getPosition().y - (heightTop * constants::Highlighter::codeScale.y) -
00037             static_cast<float>(this->linesCount - 1 - i) * this->rectSize.y
00038         );
00039         this->lines.push_back(rect);
00040     }
00041 }
```

### 7.17.3 Member Function Documentation

#### 7.17.3.1 render()

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

Definition at line 45 of file [Highlighter.cpp](#).

```
00045     {
00046     this->window->draw(this->codeSprite);
00047
00048     for (auto &i : this->toggleLines) {
00049         this->window->draw(this->lines[i]);
00050     }
00051 }
```

### 7.17.3.2 resetToggle()

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

Definition at line 53 of file [Highlighter.cpp](#).

```
00053     {
00054         this->toggleLines.clear();
00055     }
```

### 7.17.3.3 toggle()

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

Definition at line 41 of file [Highlighter.cpp](#).

```
00041     {
00042         this->toggleLines = std::move(linesList);
00043     }
```

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](#).

```
00018                                     {
00019         SINGLY,
00020         DOUBLY,
00021         CIRCULAR
00022     };
```

### 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](#).

```
00007                                     {
00008     this->window = window;
00009     this->typeLinkedList = typeLinkedList;
00010     this->highlighter = nullptr;
00011     this->delayTime = constants::LinkedList::DELAY_TIME;
00012     this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});
00013
00014     if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
00015         this->backArrow->show();
00016     else
00017         this->backArrow->hide();
00018
00019     this->createLinkedList(0);
00020 }
```

## 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;
```

```

00322
00323     sf::Vector2f newPosition(
00324         constants::NodeInfo::originNode.x + static_cast<float>(this->nodes.size()) *
constants::NodeInfo::offsetX,
00325         constants::NodeInfo::originNode.y
00326     );
00327     if (this->size) {
00328         this->nodes.back()->initArrow(
00329             NodeInfo::ArrowType::RIGHT,
00330             this->nodes.back()->getPosition(),
00331             newPosition
00332         );
00333     }
00334     this->nodes.push_back(new NodeInfo(
00335         this->window,
00336         "10",
00337         newPosition,
00338         this->typeLinkedList == TypeLinkedList::DOUBLY
00339     ));
00340     ++this->size;
00341     if (this->typeLinkedList == TypeLinkedList::DOUBLY && this->size > 1)
00342         this->nodes.back()->initArrow(
00343             NodeInfo::ArrowType::LEFT,
00344             this->nodes.back()->getPosition(),
00345             this->nodes[this->nodes.size() - 2]->getPosition()
00346         );
00347     this->backArrow->setPosition(newPosition, this->nodes[0]->getPosition());
00348     for (int i = this->size - 1; i > position; --i) {
00349         this->nodes[i]->setValue(this->nodes[i - 1]->getValue());
00350         this->nodes[i]->reInitPreVal();
00351     }
00352     this->nodes[position]->setValue(std::move(value));
00353     // std::cout << "add node to the current list " << position << " " << this->nodes[position]->getValue()
<< std::endl;
00354
00355     this->chosenNode = position;
00356     this->currentEvent = 0;
00357
00358     for (auto &e : listEvents)
00359         this->events.emplace_back(e);
00360 }

```

#### 7.18.4.2 calculateEffectivePositions()

```
void LinkedList::calculateEffectivePositions ( )
```

Definition at line 147 of file [LinkedList.cpp](#).

```

00147
00148     if (this->size < 2) return;
00149
00150     int lastInChain = 0;
00151     if (this->nodes[lastInChain]->getStatusNode() != NodeInfo::StatusNode::InChain) {
00152         lastInChain++;
00153     }
00154
00155     this->nodes[lastInChain]->setEffectivePosition(
00156         sf::Vector2f(
00157             constants::NodeInfo::originNode.x,
00158             constants::NodeInfo::originNode.y
00159         )
00160     );
00161
00162     for (int i = lastInChain + 1; i < this->size; i++){
00163         if (this->nodes[i]->getStatusNode() == NodeInfo::StatusNode::InChain) {
00164             this->nodes[i]->setEffectivePosition(
00165                 sf::Vector2f(
00166                     this->nodes[lastInChain]->getPosition().x + constants::NodeInfo::offsetX,
00167                     this->nodes[lastInChain]->getPosition().y
00168                 )
00169             );
00170             lastInChain = i;
00171         }
00172     }
00173 }

```



## 7.18.4.3 clear()

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

Definition at line 22 of file [LinkedList.cpp](#).

```
00022         {
00023     for (auto &node : this->nodes)
00024         delete node;
00025     this->nodes.clear();
00026     this->size = 0;
00027 }
```

## 7.18.4.4 createLinkedList() [1/2]

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

Definition at line 201 of file [LinkedList.cpp](#).

```
00201         {
00202     this->resetEvents();
00203     this->size = _size;
00204     for (auto &node : this->nodes)
00205         delete node;
00206     this->nodes.resize(_size);
00207     for (int i = 0; i < size; i++){
00208         this->nodes[i] = new NodeInfo(
00209             this->window,
00210             std::to_string(Random::randomInt(0, 99)),
00211             sf::Vector2f(
00212                 constants::NodeInfo::originNode.x + static_cast<float>(i) *
00213                 constants::NodeInfo::offsetX,
00214                 constants::NodeInfo::originNode.y
00215             ),
00216             this->typeLinkedList == TypeLinkedList::DOUBLY
00217         );
00218         if (i > 0){
00219             this->nodes[i - 1]->initArrow(
00220                 NodeInfo::ArrowType::RIGHT,
00221                 this->nodes[i - 1]->getPosition(),
00222                 this->nodes[i]->getPosition()
00223             );
00224             if (this->typeLinkedList == TypeLinkedList::DOUBLY)
00225                 this->nodes[i]->initArrow(
00226                     NodeInfo::ArrowType::LEFT,
00227                     this->nodes[i]->getPosition(),
00228                     this->nodes[i - 1]->getPosition()
00229                 );
00230         }
00231         if (this->size > 1)
00232             this->backArrow->setPosition(this->nodes.back()->getPosition(),
00233             this->nodes[0]->getPosition());
00234     }
```

## 7.18.4.5 createLinkedList() [2/2]

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

Definition at line 235 of file [LinkedList.cpp](#).

```
00235         {
00236     this->resetEvents();
00237     this->size = static_cast<int>(values.size());
00238     for (auto &node : this->nodes)
00239         delete node;
00240     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](#).

```
00399     {
00400     if (position < 0 || position >= this->size) return {};
00401     return this->nodes[position]->getPosition();
00402 }
```

#### 7.18.4.9 getSize()

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

Definition at line 316 of file [LinkedList.cpp](#).

```
00316     {
00317     return this->size;
00318 }
```

#### 7.18.4.10 initHighlighter()

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

Definition at line 269 of file [LinkedList.cpp](#).

```
00269     {
00270     delete this->highlighter;
00271     this->highlighter = new Highlighter(
00272         this->window,
00273         linesCount,
00274         codePath
00275     );
00276 }
```

#### 7.18.4.11 processControlMenu()

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

Definition at line 287 of file [LinkedList.cpp](#).

```
00287     {
00288     if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)
00289         return;
00290     switch (status){
00291     case ControlMenu::StatusCode::PREVIOUS:
00292         if (this->currentEvent > 0)
00293             --this->currentEvent;
00294         break;
00295     case ControlMenu::StatusCode::PAUSE:
00296         // std::cout << "PAUSE" << std::endl;
00297         break;
00298     case ControlMenu::StatusCode::PLAY:
00299         if (this->currentEvent + 1 < this->events.size()) {
00300             this->isDelay = true;
00301             this->clock.restart();
00302         }
00303     case ControlMenu::StatusCode::NEXT:
00304         if (this->currentEvent + 1 < this->events.size())
00305             ++this->currentEvent;
00306         break;
00307     default:
00308         break;
00309     }
00310 }
```

#### 7.18.4.12 render()

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

Definition at line 29 of file [LinkedList.cpp](#).

```
00029     {
00030         if (this->size > 1) {
00031             //         this->backArrow->toggleActiveColorNode();
00032             this->backArrow->render();
00033         }
00034         for (auto &node : this->nodes){
00035             node->render();
00036         }
00037     }
```

#### 7.18.4.13 renderHighlighter()

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

Definition at line 282 of file [LinkedList.cpp](#).

```
00282     {
00283         if (this->highlighter)
00284             this->highlighter->render();
00285     }
```

#### 7.18.4.14 resetEvents()

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

Definition at line 175 of file [LinkedList.cpp](#).

```
00175     {
00176         delete this->highlighter;
00177         this->highlighter = nullptr;
00178         this->currentEvent = 0;
00179         this->events.clear();
00180         this->chosenNode = 0;
00181
00182         if (this->deletedNode != -1){
00183             this->nodes.erase(this->nodes.begin() + this->deletedNode);
00184             --this->size;
00185             if (this->size && this->deletedNode == this->size)
00186                 this->nodes.back()->destroyArrow(NodeInfo::ArrowType::RIGHT);
00187             if (this->size && this->deletedNode == 0)
00188                 this->nodes[0]->destroyArrow(NodeInfo::ArrowType::LEFT);
00189         }
00190         this->deletedNode = -1;
00191
00192         for (int i = 0; i < this->size; i++){
00193             this->nodes[i]->reset();
00194             this->nodes[i]->reInitPos(i);
00195             this->nodes[i]->reInitPreVal();
00196         }
00197         if (this->size > 1)
00198             this->backArrow->setPosition(this->nodes.back()->getPosition(),
00199             this->nodes[0]->getPosition());
00199     }
```

#### 7.18.4.15 searchNode()

```
void LinkedList::searchNode (
    const std::vector< EventAnimation > & listEvents )
```

Definition at line 384 of file [LinkedList.cpp](#).

```
00384                                     {
00385     this->chosenNode = 0;
00386     this->currentEvent = 0;
00387
00388     for (auto &e : listEvents)
00389         this->events.emplace_back(e);
00390 }
```

#### 7.18.4.16 setSpeed()

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

Definition at line 312 of file [LinkedList.cpp](#).

```
00312                                     {
00313     this->speed = _speed;
00314 }
```

#### 7.18.4.17 toggleLines()

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

Definition at line 278 of file [LinkedList.cpp](#).

```
00278                                     {
00279     this->highlighter->toggle(std::move(lines));
00280 }
```

#### 7.18.4.18 update()

```
void LinkedList::update ( )
```

Definition at line 69 of file [LinkedList.cpp](#).

```
00069     {
00070         if ((int)this->events.size() && (this->isDelay or this->clock.getElapsedTime().asSeconds() >
            this->delayTime / this->speed))
00071             this->updateAnimation();
00072         this->isDelay = false;
00073     }
```

### 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,
00125 this->nodes[this->chosenNode + 1]->getPosition());
00126         if (this->chosenNode > 0)
00127             this->nodes[this->chosenNode]->updateArrows(NodeInfo::ArrowType::LEFT,
00128 this->nodes[this->chosenNode - 1]->getPosition());
00129
00130         if (event.indexBackArrow.first != -1 and event.indexBackArrow.second != -1)
00131             this->backArrow->setPosition(
00132                 this->nodes[event.indexBackArrow.first]->getPosition(),
00133                 this->nodes[event.indexBackArrow.second]->getPosition()
00134             );
00135
00136         int lastInChain = 0;
00137         if (this->nodes[lastInChain]->getStatusNode() != NodeInfo::StatusNode::InChain) {
00138             lastInChain++;
00139         }
00140         for (int i = lastInChain + 1; i < this->size; i++){
00141             if (this->nodes[i]->getStatusNode() == NodeInfo::StatusNode::InChain) {
00142                 this->nodes[lastInChain]->updateArrows(NodeInfo::ArrowType::RIGHT,
00143 this->nodes[i]->getPosition());
00144                 this->nodes[i]->updateArrows(NodeInfo::ArrowType::LEFT,
00145 this->nodes[lastInChain]->getPosition());
00146                 lastInChain = i;
00147             }
00148         }
00149     }

```

## 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](#).

```
00373 {
00374     if (position < 0 || position >= this->size) return;
00375
00376     this->nodes[position]->setValue(std::move(value));
00377     this->chosenNode = position;
00378     this->currentEvent = 0;
00379
00380     for (auto &e : listEvents)
00381         this->events.emplace_back(e);
00382 }
```

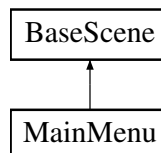
The documentation for this class was generated from the following files:

- [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 [resetActiveOptionsMenu](#) ()
- void [pollEvents](#) (sf::Event event, sf::Vector2f mousePosView)
- void [update](#) ()
- void [render](#) ()
- [Button](#) \* [getButton](#) (int index)
- [constants::MenuArray::Button](#) [getActiveOptionsMenu](#) ()

### 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](#).

```
00007
00008     this->window = window;
00009     this->typeArray = _typeArray;
00010     this->init();
00011 }
```

#### 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](#).

```
00158
00159     return this->activeCreateMode;
00160 }
```

#### 7.20.3.2 getActiveOptionsMenu()

```
constants::MenuArray::Button MenuArray::getActiveOptionsMenu ( )
```

Definition at line 154 of file [MenuArray.cpp](#).

```
00154
00155     return this->activeOptionsMenu;
00156 }
```

### 7.20.3.3 getButton()

```
Button * MenuArray::getButton (
    int index )
```

Definition at line 150 of file [MenuArray.cpp](#).

```
00150     {
00151         return this->buttons[index];
00152     }
```

### 7.20.3.4 pollEvents()

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

Definition at line 53 of file [MenuArray.cpp](#).

```
00053     {
00054         if (this->activeOptionsMenu != constants::MenuArray::Button::NONE)
00055             this->buttons[this->activeOptionsMenu]->setColor(constants::normalGray);
00056
00057         for (int i = 0; i < constants::MenuArray::BUTTON_COUNT; ++i) {
00058             if (this->buttons[i]->pollEvent(mousePosView)) {
00059                 std::cout << "Button " << i << " is clicked" << std::endl;
00060                 this->activeOptionsMenu = static_cast<constants::MenuArray::Button>(i);
00061                 this->activeAddMode = constants::MenuArray::AddMode::Textbox::NONE;
00062             }
00063         }
00064
00065         switch (this->activeOptionsMenu) {
00066             case constants::MenuArray::Button::CREATE_BUTTON:
00067                 this->pollEventCreateMode(event, mousePosView);
00068                 break;
00069             case constants::MenuArray::Button::ADD_BUTTON:
00070                 this->pollEventAddMode(event, mousePosView);
00071                 break;
00072             case constants::MenuArray::Button::DELETE_BUTTON:
00073                 this->pollEventDeleteMode(event, mousePosView);
00074                 break;
00075             case constants::MenuArray::Button::UPDATE_BUTTON:
00076                 this->pollEventUpdateMode(event, mousePosView);
00077                 break;
00078             case constants::MenuArray::Button::SEARCH_BUTTON:
00079                 this->pollEventSearchMode(event, mousePosView);
00080                 break;
00081             case constants::MenuArray::Button::ALLOCATE_BUTTON:
00082                 this->pollEventAllocateMode(event, mousePosView);
00083                 break;
00084             case constants::MenuArray::Button::NONE:
00085                 break;
00086         }
00087     }
```

### 7.20.3.5 render()

```
void MenuArray::render ( )
```

Definition at line 121 of file [MenuArray.cpp](#).

```
00121     {
00122         for (Button* button : this->buttons) {
00123             button->render();
00124         }
00125
00126         switch (this->activeOptionsMenu) {
00127             case constants::MenuArray::Button::CREATE_BUTTON:
00128                 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 resetActiveOptionsMenu()

```
void MenuArray::resetActiveOptionsMenu ( )
```

Definition at line 48 of file [MenuArray.cpp](#).

```

00048     {
00049         this->activeOptionsMenu = 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->activeOptionsMenu != constants::MenuArray::Button::NONE)
00091             this->buttons[this->activeOptionsMenu]->setColor(constants::clickGreen);
00092
00093         for (Button* button : this->buttons) {
00094             button->update();
00095         }
00096
00097         switch (this->activeOptionsMenu) {
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 [resetActiveOptionsMenu](#) ()
- void [resetActiveOptionsMenuOnly](#) ()
- void [pollEvents](#) (sf::Event event, sf::Vector2f mousePosView)
- void [update](#) ()
- void [render](#) ()
- [Button \\* getButton](#) (int index)
- [constants::MenuDataStructure::Button getActiveOptionsMenu](#) ()

### 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](#).

```
00007                                     {
00008     this->window = window;
00009     this->init();
00010 }
```

### 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](#).

```
00094                                     {
00095     return this->activeCreateMode;
00096 }
```

### 7.21.3.2 getActiveOptionsMenu()

```
constants::MenuDataStructure::Button MenuDataStructure::getActiveOptionsMenu ( )
```

Definition at line 90 of file [MenuDataStructure.cpp](#).

```
00090                                     {
00091     return this->activeOptionsMenu;
00092 }
```

### 7.21.3.3 getButton()

```
Button * MenuDataStructure::getButton (
    int index )
```

Definition at line 86 of file [MenuDataStructure.cpp](#).

```
00086                                     {
00087     return this->buttons[index];
00088 }
```

### 7.21.3.4 pollEvents()

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

Definition at line 41 of file [MenuDataStructure.cpp](#).

```
00041         {
00042     if (this->activeOptionMenu != constants::MenuDataStructure::Button::NONE)
00043         this->buttons[this->activeOptionMenu]->setColor(constants::normalGray);
00044
00045     for (int i = 0; i < constants::MenuDataStructure::BUTTON_COUNT; i++) {
00046         if (this->buttons[i]->pollEvent(mousePosView)) {
00047             std::cout << "Button " << i << " is clicked" << std::endl;
00048             this->activeOptionMenu = static_cast<constants::MenuDataStructure::Button>(i);
00049         }
00050     }
00051
00052     if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
00053         this->pollEventCreateMode(event, mousePosView);
00054     } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
00055         this->pollEventPushMode(event, mousePosView);
00056     }
00057 }
```

### 7.21.3.5 render()

```
void MenuDataStructure::render ( )
```

Definition at line 74 of file [MenuDataStructure.cpp](#).

```
00074         {
00075     for (Button* button : this->buttons) {
00076         button->render();
00077     }
00078
00079     if (this->activeOptionMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
00080         this->renderCreateMode();
00081     } else if (this->activeOptionMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
00082         this->renderPushMode();
00083     }
00084 }
```

### 7.21.3.6 resetActiveOptionsMenu()

```
void MenuDataStructure::resetActiveOptionsMenu ( )
```

Definition at line 239 of file [MenuDataStructure.cpp](#).

```
00239         {
00240     this->activeOptionMenu = constants::MenuDataStructure::Button::NONE;
00241     this->activeCreateMode = constants::MenuDataStructure::CreateMode::Button::NONE;
00242 }
```

### 7.21.3.7 resetActiveOptionsMenuOnly()

```
void MenuDataStructure::resetActiveOptionsMenuOnly ( )
```

Definition at line 244 of file [MenuDataStructure.cpp](#).

```
00244         {
00245     this->activeOptionMenu = constants::MenuDataStructure::Button::NONE;
00246 }
```



### 7.21.3.8 update()

```
void MenuDataStructure::update ( )
```

Definition at line 59 of file [MenuDataStructure.cpp](#).

```
00059     {
00060         if (this->activeOptionsMenu < constants::MenuDataStructure::Button::POP_BUTTON)
00061             this->buttons[this->activeOptionsMenu]->setColor(constants::clickGreen);
00062
00063         for (Button* button : this->buttons) {
00064             button->update();
00065         }
00066
00067         if (this->activeOptionsMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
00068             this->updateCreateMode();
00069         } else if (this->activeOptionsMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
00070             this->updatePushMode();
00071         }
00072     }
```

## 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 [resetActiveOptionsMenu \(\)](#)
- void [pollEvents \(sf::Event event, sf::Vector2f mousePosView\)](#)
- void [update \(\)](#)
- void [render \(\)](#)
- [Button \\* getButton \(int index\)](#)
- [constants::MenuLinkedList::Button getActiveOptionsMenu \(\)](#)

## 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 activeOptionsMenu`
- `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](#).

```
00039                                     {
00040     this->window = window;
00041     this->init();
00042 }
```

#### 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](#).

```
00242                                     {
00243     return this->activeCreateMode;
00244 }
```

### 7.22.3.2 getActiveOptionsMenu()

`constants::MenuLinkedList::Button MenuLinkedList::getActiveOptionsMenu ( )`

Definition at line 246 of file [MenuLinkedList.cpp](#).

```
00246                                     {
00247     return this->activeOptionsMenu;
00248 }
```

### 7.22.3.3 getButton()

`Button * MenuLinkedList::getButton (
 int index )`

Definition at line 132 of file [MenuLinkedList.cpp](#).

```
00132                                     {
00133     return this->buttons[index];
00134 }
```

### 7.22.3.4 init()

`void MenuLinkedList::init ( ) [protected]`

Definition at line 7 of file [MenuLinkedList.cpp](#).

```
00007     {
00008         this->initButtons();
00009         this->initCreateMode();
00010         this->initAddMode();
00011         this->initDeleteMode();
00012         this->initUpdateMode();
00013         this->initSearchMode();
00014
00015         this->activeOptionsMenu = constants::MenuLinkedList::Button::NONE;
00016     }
```

## 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](#).

```
00333     {
00334         // init stuff for update mode
00335         this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::Textbox::NONE;
00336         for (int i = 0; i < constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT; i++) {
00337             sf::Vector2f position = sf::Vector2f(
00338                 this->buttons[3]->getPosition().x + (constants::optionButtonSize.x +
constants::distance2ModeButtons),
00339                 this->buttons[3]->getPosition().y
00340             );
00341             this->updateTextbox[i] = new CustomTextbox{
00342                 this->window,
00343                 position,
00344                 20,
00345                 constants::MenuLinkedList::UpdateMode::TEXTBOX_NAMES[i],
00346                 constants::MenuLinkedList::UpdateMode::TEXTBOX_LENGTH[i],
00347             };
00348             this->updateModeValue[i] = "None";
00349         }
00350     }
```

## 7.22.3.11 pollEventAddMode()

```
void MenuLinkedList::pollEventAddMode (
    sf::Event event,
    sf::Vector2f mousePosView ) [protected]
```

Definition at line 268 of file [MenuLinkedList.cpp](#).

```
00268     {
00269         if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
00270             this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;
00271
00272         this->addTextbox[this->activeAddMode]->pollEvent(event, mousePosView);
00273     }
```

## 7.22.3.12 pollEventCreateMode()

```
void MenuLinkedList::pollEventCreateMode (
    sf::Event event,
    sf::Vector2f mousePosView ) [protected]
```

Definition at line 176 of file [MenuLinkedList.cpp](#).

```
00176     {
00177         if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
00178             this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);
00179
00180         for (int i = 0; i < constants::MenuLinkedList::CreateMode::BUTTON_COUNT; i++) {
00181             if (this->subCreateMode[i]->pollEvent(mousePosView)) {
00182                 this->activeCreateMode = static_cast<constants::MenuLinkedList::CreateMode::Button>(i);
00183                 if (i == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON)
00184                     this->isOpenFileDialog = true;
00185                 std::cout << "Button " << i << " is clicked" << std::endl;
00186             }
00187         }
00188
00189         // this->testTextbox->pollEvent(event);
00190         if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
00191             this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
00192     }
```

### 7.22.3.13 pollEventDeleteMode()

```
void MenuLinkedList::pollEventDeleteMode (
    sf::Event event,
    sf::Vector2f mousePosView ) [protected]
```

Definition at line 311 of file [MenuLinkedList.cpp](#).

```
00311 {
00312     this->deleteTextbox->pollEvent(event, mousePosView);
00313 }
```

### 7.22.3.14 pollEvents()

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

Definition at line 44 of file [MenuLinkedList.cpp](#).

```
00044 {
00045     if (this->activeOptionsMenu != constants::MenuLinkedList::Button::NONE)
00046         this->buttons[this->activeOptionsMenu]->setColor(constants::normalGray);
00047
00048     for (int i = 0; i < constants::MenuLinkedList::BUTTON_COUNT; i++) {
00049         if (this->buttons[i]->pollEvent(mousePosView)) {
00050             std::cout << "Button " << i << " is clicked" << std::endl;
00051             this->activeOptionsMenu = static_cast<constants::MenuLinkedList::Button>(i);
00052             this->activeAddMode = constants::MenuLinkedList::AddMode::Textbox::NONE;
00053         }
00054     }
00055
00056     switch (this->activeOptionsMenu) {
00057         case constants::MenuLinkedList::Button::CREATE_BUTTON:
00058             this->pollEventCreateMode(event, mousePosView);
00059             break;
00060         case constants::MenuLinkedList::Button::ADD_BUTTON:
00061             this->pollEventAddMode(event, mousePosView);
00062             break;
00063         case constants::MenuLinkedList::Button::DELETE_BUTTON:
00064             this->pollEventDeleteMode(event, mousePosView);
00065             break;
00066         case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00067             this->pollEventUpdateMode(event, mousePosView);
00068             break;
00069         case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00070             this->pollEventSearchMode(event, mousePosView);
00071             break;
00072         case constants::MenuLinkedList::Button::NONE:
00073             break;
00074     }
00075 }
```

### 7.22.3.15 pollEventSearchMode()

```
void MenuLinkedList::pollEventSearchMode (
    sf::Event event,
    sf::Vector2f mousePosView ) [protected]
```

Definition at line 394 of file [MenuLinkedList.cpp](#).

```
00394 {
00395     this->searchTextbox->pollEvent(event, mousePosView);
00396 }
```



### 7.22.3.16 pollEventUpdateMode()

```
void MenuLinkedList::pollEventUpdateMode (
    sf::Event event,
    sf::Vector2f mousePosView ) [protected]
```

Definition at line 351 of file [MenuLinkedList.cpp](#).

```
00351 {
00352     if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
00353         this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;
00354
00355     this->updateTextbox[this->activeUpdateMode]->pollEvent(event, mousePosView);
00356 }
```

### 7.22.3.17 render()

```
void MenuLinkedList::render ( )
```

Definition at line 106 of file [MenuLinkedList.cpp](#).

```
00106 {
00107     for (Button* button : this->buttons) {
00108         button->render();
00109     }
00110
00111     switch (this->activeOptionsMenu) {
00112         case constants::MenuLinkedList::Button::CREATE_BUTTON:
00113             this->renderCreateMode();
00114             break;
00115         case constants::MenuLinkedList::Button::ADD_BUTTON:
00116             this->renderAddMode();
00117             break;
00118         case constants::MenuLinkedList::Button::DELETE_BUTTON:
00119             this->renderDeleteMode();
00120             break;
00121         case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00122             this->renderUpdateMode();
00123             break;
00124         case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00125             this->renderSearchMode();
00126             break;
00127         case constants::MenuLinkedList::Button::NONE:
00128             break;
00129     }
00130 }
```

### 7.22.3.18 renderAddMode()

```
void MenuLinkedList::renderAddMode ( ) [protected]
```

Definition at line 293 of file [MenuLinkedList.cpp](#).

```
00293 {
00294     this->addTextbox[this->activeAddMode]->render();
00295 }
```

### 7.22.3.19 renderCreateMode()

void MenuLinkedList::renderCreateMode ( ) [protected]

Definition at line 232 of file [MenuLinkedList.cpp](#).

```
00232     {
00233         for (Button* button : this->subCreateMode) {
00234             button->render();
00235         }
00236     }
00237     //     this->testTextbox->render();
00238     if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
00239         this->createTextbox[this->activeCreateMode]->render();
00240 }
```

### 7.22.3.20 renderDeleteMode()

void MenuLinkedList::renderDeleteMode ( ) [protected]

Definition at line 329 of file [MenuLinkedList.cpp](#).

```
00329     {
00330         this->deleteTextbox->render();
00331     }
```

### 7.22.3.21 renderSearchMode()

void MenuLinkedList::renderSearchMode ( ) [protected]

Definition at line 412 of file [MenuLinkedList.cpp](#).

```
00412     {
00413         this->searchTextbox->render();
00414     }
```

### 7.22.3.22 renderUpdateMode()

void MenuLinkedList::renderUpdateMode ( ) [protected]

Definition at line 376 of file [MenuLinkedList.cpp](#).

```
00376     {
00377         this->updateTextbox[this->activeUpdateMode]->render();
00378     }
```

### 7.22.3.23 resetActiveOptionsMenu()

void MenuLinkedList::resetActiveOptionsMenu ( )

Definition at line 136 of file [MenuLinkedList.cpp](#).

```
00136     {
00137         this->activeOptionsMenu = constants::MenuLinkedList::Button::NONE;
00138         this->activeCreateMode = constants::MenuLinkedList::CreateMode::Button::NONE;
00139     }
```

## 7.22.3.24 update()

```
void MenuLinkedList::update ( )
```

Definition at line 77 of file [MenuLinkedList.cpp](#).

```
00077         {
00078     if (this->activeOptionsMenu != constants::MenuLinkedList::Button::NONE)
00079         this->buttons[this->activeOptionsMenu]->setColor(constants::clickGreen);
00080
00081     for (Button* button : this->buttons) {
00082         button->update();
00083     }
00084
00085     switch (this->activeOptionsMenu) {
00086     case constants::MenuLinkedList::Button::CREATE_BUTTON:
00087         this->updateCreateMode();
00088         break;
00089     case constants::MenuLinkedList::Button::ADD_BUTTON:
00090         this->updateAddMode();
00091         break;
00092     case constants::MenuLinkedList::Button::DELETE_BUTTON:
00093         this->updateDeleteMode();
00094         break;
00095     case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00096         this->updateUpdateMode();
00097         break;
00098     case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00099         this->updateSearchMode();
00100         break;
00101     case constants::MenuLinkedList::Button::NONE:
00102         break;
00103     }
00104 }
```

## 7.22.3.25 updateAddMode()

```
void MenuLinkedList::updateAddMode ( ) [protected]
```

Definition at line 274 of file [MenuLinkedList.cpp](#).

```
00274         {
00275     if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
00276         this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;
00277
00278     this->addTextbox[this->activeAddMode]->update();
00279
00280     std::string inputUser = this->addTextbox[this->activeAddMode]->getTextString();
00281     // check if input is number
00282     bool isValid = true;
00283     for (char i : inputUser)
00284         if (!std::isdigit(i))
00285             isValid = false;
00286     if (isValid && inputUser != "None") {
00287         this->addModeValue[this->activeAddMode] = inputUser;
00288         std::cout << inputUser << std::endl;
00289         this->addTextbox[this->activeAddMode]->resetInput();
00290         this->activeAddMode =
00291         static_cast<constants::MenuLinkedList::AddMode::Textbox>(!this->activeAddMode);
00292     }
```

## 7.22.3.26 updateCreateMode()

```
void MenuLinkedList::updateCreateMode ( ) [protected]
```

Definition at line 193 of file [MenuLinkedList.cpp](#).

```
00193         {
00194     if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
```

```

00195         this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);
00196
00197     for (Button* button : this->subCreateMode) {
00198         button->update();
00199     }
00200
00201     // this->testTextbox->update();
00202     if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT) {
00203         this->createTextbox[this->activeCreateMode]->update();
00204         std::string inputUser = this->createTextbox[this->activeCreateMode]->getTextString();
00205         if (inputUser != "None") {
00206             std::cout << inputUser << std::endl;
00207             this->createTextbox[this->activeCreateMode]->resetInput();
00208         }
00209         this->createModeValue[this->activeCreateMode] = inputUser;
00210     } else if (this->activeCreateMode == constants::MenuLinkedList::CreateMode::FILE_BUTTON) {
00211         if (this->isOpenFileDialog) {
00212             auto f = pfd::open_file("Choose files to read", pfd::path::home(),
00213                                   {"Text Files (.txt .text)", "*.txt *.text",
00214                                    "All Files", "*"});
00215
00216             // wait for the user to select a file unless the window will be not responsive
00217             while (!f.ready(100)) {
00218                 sf::Event event{};
00219                 this->window->pollEvent(event);
00220             }
00221
00222             if (!f.result().empty()) {
00223                 std::ifstream file(f.result()[0]);
00224                 std::string line;
00225                 file >> line;
00226                 this->createModeValue[this->activeCreateMode] = line;
00227             }
00228         }
00229         this->isOpenFileDialog = false;
00230     }
00231 }

```

### 7.22.3.27 updateDeleteMode()

void MenuLinkedList::updateDeleteMode ( ) [protected]

Definition at line 314 of file [MenuLinkedList.cpp](#).

```

00314     {
00315         this->deleteTextbox->update();
00316
00317         std::string inputUser = this->deleteTextbox->getTextString();
00318         // check if input is number
00319         bool isValid = true;
00320         for (char i : inputUser)
00321             if (!std::isdigit(i))
00322                 isValid = false;
00323         if (isValid && inputUser != "None") {
00324             this->deleteModeValue = inputUser;
00325             std::cout << inputUser << std::endl;
00326             this->deleteTextbox->resetInput();
00327         }
00328     }

```

### 7.22.3.28 updateSearchMode()

void MenuLinkedList::updateSearchMode ( ) [protected]

Definition at line 397 of file [MenuLinkedList.cpp](#).

```

00397     {
00398         this->searchTextbox->update();
00399
00400         std::string inputUser = this->searchTextbox->getTextString();
00401         // check if input is number
00402         bool isValid = true;

```

```

00403     for (char i : inputUser)
00404         if (!std::isdigit(i))
00405             isValid = false;
00406     if (isValid && inputUser != "None") {
00407         this->searchModeValue = inputUser;
00408         std::cout << inputUser << std::endl;
00409         this->searchTextbox->resetInput();
00410     }
00411 }

```

### 7.22.3.29 updateUpdateMode()

```
void MenuLinkedList::updateUpdateMode ( ) [protected]
```

Definition at line 357 of file [MenuLinkedList.cpp](#).

```

00357     {
00358         if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
00359             this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;
00360
00361         this->updateTextbox[this->activeUpdateMode]->update();
00362
00363         std::string inputUser = this->updateTextbox[this->activeUpdateMode]->getTextString();
00364         // check if input is number
00365         bool isValid = true;
00366         for (char i : inputUser)
00367             if (!std::isdigit(i))
00368                 isValid = false;
00369         if (isValid && inputUser != "None") {
00370             this->updateModeValue[this->activeUpdateMode] = inputUser;
00371             std::cout << inputUser << std::endl;
00372             this->updateTextbox[this->activeUpdateMode]->resetInput();
00373             this->activeUpdateMode =
00374                 static_cast<constants::MenuLinkedList::UpdateMode::Textbox>(!this->activeUpdateMode);
00375         }
00376     }

```

## 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 activeOptionsMenu

```
constants::MenuLinkedList::Button MenuLinkedList::activeOptionsMenu [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](#).

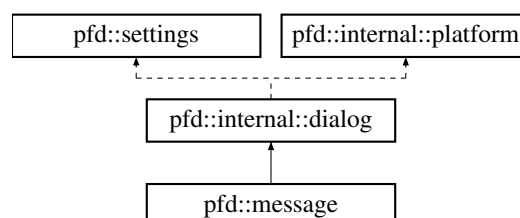
The documentation for this class was generated from the following files:

- [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`.

```
01600 {
01601 #if _WIN32
01602     // Use MB_SYSTEMMODAL rather than MB_TOPMOST to ensure the message window is brought
01603     // to front. See https://github.com/samhocevar/portable-file-dialogs/issues/52
01604     UINT style = MB_SYSTEMMODAL;
01605     switch (_icon)
01606     {
01607         case icon::warning: style |= MB_ICONWARNING; break;
01608         case icon::error: style |= MB_ICONERROR; break;
01609         case icon::question: style |= MB_ICONQUESTION; break;
01610         /* case icon::info: */ default: style |= MB_ICONINFORMATION; break;
01611     }
01612     switch (_choice)
01613     {
01614         case choice::ok_cancel: style |= MB_OKCANCEL; break;
01615         case choice::yes_no: style |= MB_YESNO; break;
01616         case choice::yes_no_cancel: style |= MB_YESNOCANCEL; break;
01617         case choice::retry_cancel: style |= MB_RETRYCANCEL; break;
01618         case choice::abort_retry_ignore: style |= MB_ABORTRETRYIGNORE; break;
01619         /* case choice::ok: */ default: style |= MB_OK; break;
01620     }
01621     m_mappings[IDCANCEL] = button::cancel;
01622     m_mappings[IDOK] = button::ok;
01623     m_mappings[IDYES] = button::yes;
01624     m_mappings[IDNO] = button::no;
01625     m_mappings[IDABORT] = button::abort;
01626     m_mappings[IDRETRY] = button::retry;
01627     m_mappings[IDIGNORE] = button::ignore;
01628     m_async->start_func([text, title, style](int* exit_code) -> std::string
01629     {
01630         auto wtext = internal::str2wstr(text);
01631         auto wtitle = internal::str2wstr(title);
01632         // Apply new visual style (required for all Windows versions)
01633         new_style_context ctx;
01634         *exit_code = MessageBoxW(GetActiveWindow(), wtext.c_str(), wtitle.c_str(), style);
01635         return "";
01636     });
01637 #elif __EMSCRIPTEN__
01638     std::string full_message;
01639     switch (_icon)
01640     {
01641         case icon::warning: full_message = ""; break;
01642         case icon::error: full_message = ""; break;
01643         case icon::question: full_message = ""; break;
```

```

01648     /* case icon::info: */ default: full_message = ""; break;
01649 }
01650
01651 full_message += ' ' + title + "\n\n" + text;
01652
01653 // This does not really start an async task; it just passes the
01654 // EM_ASM_INT return value to a fake start() function.
01655 m_async->start(EM_ASM_INT(
01656 {
01657     if ($1)
01658         return window.confirm(UTF8ToString($0)) ? 0 : -1;
01659     alert(UTF8ToString($0));
01660     return 0;
01661 }, full_message.c_str(), _choice == choice::ok_cancel));
01662 #else
01663     auto command = desktop_helper();
01664
01665     if (is_osascript())
01666     {
01667         std::string script = "display dialog " + osascript_quote(text) +
01668                             " with title " + osascript_quote(title);
01669         auto if_cancel = button::cancel;
01670         switch (_choice)
01671         {
01672             case choice::ok_cancel:
01673                 script += "buttons {\\"OK\\", \\"Cancel\\"}"
01674                         " default button \\"OK\\" "
01675                         " cancel button \\"Cancel\\"";
01676                 break;
01677             case choice::yes_no:
01678                 script += "buttons {\\"Yes\\", \\"No\\"}"
01679                         " default button \\"Yes\\" "
01680                         " cancel button \\"No\\"";
01681                 if_cancel = button::no;
01682                 break;
01683             case choice::yes_no_cancel:
01684                 script += "buttons {\\"Yes\\", \\"No\\", \\"Cancel\\"}"
01685                         " default button \\"Yes\\" "
01686                         " cancel button \\"Cancel\\"";
01687                 break;
01688             case choice::retry_cancel:
01689                 script += "buttons {\\"Retry\\", \\"Cancel\\"}"
01690                         " default button \\"Retry\\" "
01691                         " cancel button \\"Cancel\\"";
01692                 break;
01693             case choice::abort_retry_ignore:
01694                 script += "buttons {\\"Abort\\", \\"Retry\\", \\"Ignore\\"}"
01695                         " default button \\"Abort\\" "
01696                         " cancel button \\"Retry\\"";
01697                 if_cancel = button::retry;
01698                 break;
01699             case choice::ok: default:
01700                 script += "buttons {\\"OK\\"}"
01701                         " default button \\"OK\\" "
01702                         " cancel button \\"OK\\"";
01703                 if_cancel = button::ok;
01704                 break;
01705         }
01706         m_mappings[1] = if_cancel;
01707         m_mappings[256] = if_cancel; // XXX: I think this was never correct
01708         script += " with icon ";
01709         switch (_icon)
01710         {
01711             #define PFD_OSX_ICON(n) "alias ((path to library folder from system domain) as text " \
01712                                     "& \\"CoreServices:CoreTypes.bundle:Contents:Resources:" n ".icns\\")"
01713             case icon::info: default: script += PFD_OSX_ICON("ToolBarInfo"); break;
01714             case icon::warning: script += "caution"; break;
01715             case icon::error: script += "stop"; break;
01716             case icon::question: script += PFD_OSX_ICON("GenericQuestionMarkIcon"); break;
01717             #undef PFD_OSX_ICON
01718         }
01719
01720         command.push_back("-e");
01721         command.push_back(script);
01722     }
01723     else if (is_zenity())
01724     {
01725         switch (_choice)
01726         {
01727             case choice::ok_cancel:
01728                 command.insert(command.end(), { "--question", "--cancel-label=Cancel",
01729                 "--ok-label=OK" }); break;
01730             case choice::yes_no:
01731                 // Do not use standard --question because it causes No to return -1,
01732                 // which is inconsistent with the Yes/No/Cancel mode below.
01733                 command.insert(command.end(), { "--question", "--switch", "--extra-button=No",
01734                 "--extra-button=Yes" }); break;

```

```

01733         case choice::yes_no_cancel:
01734             command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
"--extra-button=No", "--extra-button=Yes" }); break;
01735         case choice::retry_cancel:
01736             command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
"--extra-button=Retry" }); break;
01737         case choice::abort_retry_ignore:
01738             command.insert(command.end(), { "--question", "--switch", "--extra-button=Ignore",
"--extra-button=Abort", "--extra-button=Retry" }); break;
01739         case choice::ok:
01740             default:
01741                 switch (_icon)
01742                 {
01743                     case icon::error: command.push_back("--error"); break;
01744                     case icon::warning: command.push_back("--warning"); break;
01745                     default: command.push_back("--info"); break;
01746                 }
01747             }
01748
01749             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) });
01754     }
01755     else if (is_kdialog())
01756     {
01757         if (_choice == choice::ok)
01758         {
01759             switch (_icon)
01760             {
01761                 case icon::error: command.push_back("--error"); break;
01762                 case icon::warning: command.push_back("--sorry"); break;
01763                 default: command.push_back("--msgbox"); break;
01764             }
01765         }
01766         else
01767         {
01768             std::string flag = "--";
01769             if (_icon == icon::warning || _icon == icon::error)
01770                 flag += "warning";
01771             flag += "yesno";
01772             if (_choice == choice::yes_no_cancel)
01773                 flag += "cancel";
01774             command.push_back(flag);
01775             if (_choice == choice::yes_no || _choice == choice::yes_no_cancel)
01776             {
01777                 m_mappings[0] = button::yes;
01778                 m_mappings[256] = button::no;
01779             }
01780         }
01781
01782         command.push_back(text);
01783         command.push_back("--title");
01784         command.push_back(title);
01785
01786         // Must be after the above part
01787         if (_choice == choice::ok_cancel)
01788             command.insert(command.end(), { "--yes-label", "OK", "--no-label", "Cancel" });
01789     }
01790
01791     if (flags(flag::is_verbose))
01792         std::cerr << "pfd: " << command << std::endl;
01793
01794     m_async->start_process(command);
01795 #endif
01796 }

```

## 7.23.3 Member Function Documentation

### 7.23.3.1 result()

`button pfd::message::result ( ) [inline]`

Definition at line 1798 of file [FileDialog.h](#).

```

01799     {
01800         int exit_code;
01801         auto ret = m_async->result(&exit_code);
01802         // osascript will say "button returned:Cancel\n"
01803         // and others will just say "Cancel\n"
01804         if (internal::ends_with(ret, "Cancel\n"))
01805             return button::cancel;
01806         if (internal::ends_with(ret, "OK\n"))
01807             return button::ok;
01808         if (internal::ends_with(ret, "Yes\n"))
01809             return button::yes;
01810         if (internal::ends_with(ret, "No\n"))
01811             return button::no;
01812         if (internal::ends_with(ret, "Abort\n"))
01813             return button::abort;
01814         if (internal::ends_with(ret, "Retry\n"))
01815             return button::retry;
01816         if (internal::ends_with(ret, "Ignore\n"))
01817             return button::ignore;
01818         if (m_mappings.count(exit_code) != 0)
01819             return m_mappings[exit_code];
01820         return exit_code == 0 ? button::ok : button::cancel;
01821     }

```

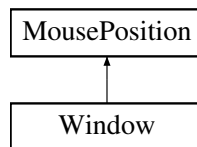
The documentation for this class was generated from the following file:

- 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](#).

```
00007     {
00008     this->mousePos = sf::Mouse::getPosition(*this->relativeWindow);
00009     this->mousePosView = this->relativeWindow->mapPixelToCoords(this->mousePos);
00010 }
```

## 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](#).

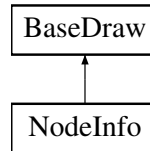
The documentation for this class was generated from the following files:

- [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](#).

```
00014 {
00015     LEFT,
00016     RIGHT
00017 };
```

### 7.25.2.2 StatusNode

```
enum class NodeInfo::StatusNode [strong]
```

#### Enumerator

InChain	
OutChain	
Visible	

Definition at line 19 of file [NodeInfo.hpp](#).



```

00019         {
00020             InChain,
00021             OutChain,
00022             Visible
00023         };

```

### 7.25.2.3 TypeNode

```
enum class NodeInfo::TypeNode [strong]
```

#### Enumerator

Normal	
Outside	
Effective	

Definition at line 25 of file [NodeInfo.hpp](#).

```

00025         {
00026             Normal,
00027             Outside,
00028             Effective
00029         };

```

## 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](#).

```

00007                                     :
    BaseDraw(window) {
00008         this->values[0] = value;
00009         this->values[1] = value;
00010
00011         this->positions[(int)TypeNode::Normal] = position;
00012         this->positions[(int)TypeNode::Effective] = position;
00013         this->positions[(int)TypeNode::Outside] = sf::Vector2f(
00014             position.x,
00015             position.y + constants::NodeInfo::offsetY
00016         );
00017
00018         this->isDLL = _isDLL;
00019
00020         this->statusNode = StatusNode::InChain;
00021
00022         this->node = new SingleNode(window, std::move(value), this->positions[(int)TypeNode::Normal]);
00023
00024         for (auto &arrow : this->arrows)
00025             arrow[(int)ArrowType::LEFT] = arrow[(int)ArrowType::RIGHT] = nullptr;
00026
00027         this->isPrintPreVal = this->isPrintNormal = false;
00028
00029         this->title.setFont(this->node->font);
00030         this->title.setCharacterSize(constants::TitleNode::fontSize);
00031         this->title.setFillColor(constants::titleGreen);
00032         this->title.setString("");
00033     }

```

### 7.25.3.2 ~NodeInfo()

NodeInfo::~~NodeInfo ( )

Definition at line 193 of file [NodeInfo.cpp](#).

```
00193     {
00194         delete this->node;
00195         for (auto & arrow : this->arrows) {
00196             for (auto & j : arrow) {
00197                 delete j;
00198             }
00199         }
00200 }
```

## 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](#).

```
00231     {
00232         if (this->arrows[0][(int)type])
00233             delete this->arrows[0][(int)type];
00234         if (this->arrows[1][(int)type])
00235             delete this->arrows[1][(int)type];
00236         this->arrows[0][(int)type] = nullptr;
00237         this->arrows[1][(int)type] = nullptr;
00238 }
```

### 7.25.4.2 getPosition()

sf::Vector2f NodeInfo::getPosition ( )

Definition at line 94 of file [NodeInfo.cpp](#).

```
00094     {
00095         this->updateNode(); // ?
00096         return this->node->getPosition();
00097 }
```

### 7.25.4.3 getStatusNode()

[NodeInfo::StatusNode](#) NodeInfo::getStatusNode ( )

Definition at line 164 of file [NodeInfo.cpp](#).

```
00164     {
00165         return this->statusNode;
00166 }
```

#### 7.25.4.4 getValue()

```
std::string NodeInfo::getValue ( )
```

Definition at line 206 of file [NodeInfo.cpp](#).

```
00206     {
00207     return this->values[0];
00208 }
```

#### 7.25.4.5 hide()

```
void NodeInfo::hide (
    NodeInfo::ArrowType type )
```

Definition at line 179 of file [NodeInfo.cpp](#).

```
00179     {
00180     if (this->arrows[0][(int)type])
00181         this->arrows[0][(int)type]->hide();
00182     if (this->arrows[1][(int)type])
00183         this->arrows[1][(int)type]->hide();
00184 }
```

#### 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](#).

```
00054     {
00055     this->arrows[1][(int)type] = new Arrow(this->window, start, end);
00056     this->arrows[1][(int)type]->setMid();
00057     this->arrows[0][(int)type] = new Arrow(this->window, start, end);
00058 }
```

#### 7.25.4.7 reInitPos()

```
void NodeInfo::reInitPos (
    int index )
```

Definition at line 99 of file [NodeInfo.cpp](#).

```
00099     {
00100     this->positions[(int)TypeNode::Normal] = sf::Vector2f(
00101         constants::NodeInfo::originNode.x + static_cast<float>(index) *
00102         constants::NodeInfo::offsetX,
00103         constants::NodeInfo::originNode.y
00104     );
00105     this->positions[(int)TypeNode::Outside] = sf::Vector2f(
00106         this->positions[(int)TypeNode::Effective].x,
00107         this->positions[(int)TypeNode::Effective].y + constants::NodeInfo::offsetY
00108     );
00109 }
```

#### 7.25.4.8 reInitPreVal()

```
void NodeInfo::reInitPreVal ( )
```

Definition at line 160 of file [NodeInfo.cpp](#).

```
00160         {
00161             this->values[1] = this->values[0];
00162         }
```

#### 7.25.4.9 render()

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

Implements [BaseDraw](#).

Definition at line 35 of file [NodeInfo.cpp](#).

```
00035         {
00036             if (this->statusNode == StatusNode::Visible)
00037                 return;
00038
00039             if (this->isDLL && this->statusNode == StatusNode::InChain) {
00040                 if (this->arrows[1] [(int) ArrowType::LEFT])
00041                     this->arrows[1] [(int) ArrowType::LEFT]->render();
00042                 if (this->arrows[1] [(int) ArrowType::RIGHT])
00043                     this->arrows[1] [(int) ArrowType::RIGHT]->render();
00044             } else {
00045                 if (this->arrows[0] [(int) ArrowType::LEFT])
00046                     this->arrows[0] [(int) ArrowType::LEFT]->render();
00047                 if (this->arrows[0] [(int) ArrowType::RIGHT])
00048                     this->arrows[0] [(int) ArrowType::RIGHT]->render();
00049             }
00050             this->node->render();
00051             this->window->draw(this->title);
00052         }
```

#### 7.25.4.10 reset()

```
void NodeInfo::reset ( )
```

Definition at line 82 of file [NodeInfo.cpp](#).

```
00082         {
00083             this->resetColorNode();
00084             this->resetColorArrow(ArrowType::LEFT);
00085             this->resetColorArrow(ArrowType::RIGHT);
00086             this->resetTitle();
00087             this->isPrintNormal = this->isPrintPreVal = false;
00088             this->statusNode = StatusNode::InChain;
00089             this->show(ArrowType::LEFT);
00090             this->show(ArrowType::RIGHT);
00091         }
```

#### 7.25.4.11 resetColorArrow()

```
void NodeInfo::resetColorArrow (
    NodeInfo::ArrowType type )
```

Definition at line 75 of file [NodeInfo.cpp](#).

```
00075         {
00076             if (this->arrows[0] [(int) type])
00077                 this->arrows[0] [(int) type]->resetColor();
00078             if (this->arrows[1] [(int) type])
00079                 this->arrows[1] [(int) type]->resetColor();
00080         }
```

#### 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](#).

```
00152                                     {
00153     if (this->arrows[0][ (int)type])
00154         this->arrows[0][ (int)type]->setPositions(this->node->getPosition(), end, false);
00155
00156     if (this->arrows[1][ (int)type])
00157         this->arrows[1][ (int)type]->setPositions(this->node->getPosition(), end, true);
00158 }
```

### 7.25.4.27 updateNode()

```
void NodeInfo::updateNode ( )
```

Definition at line 131 of file [NodeInfo.cpp](#).

```
00131     {
00132     if (this->statusNode == StatusNode::Visible)
00133         return;
00134
00135     if (this->statusNode == StatusNode::InChain) {
00136         if (this->isPrintNormal) {
00137             this->node->setPosition(this->positions[ (int)TypeNode::Normal]);
00138         } else {
00139             this->node->setPosition(this->positions[ (int)TypeNode::Effective]);
00140         }
00141     } else {
00142         this->node->setPosition(this->positions[ (int)TypeNode::Outside]);
00143     }
00144
00145     if (this->isPrintPreVal) {
00146         this->node->setText(this->values[1]);
00147     } else {
00148         this->node->setText(this->values[0]);
00149     }
00150 }
```

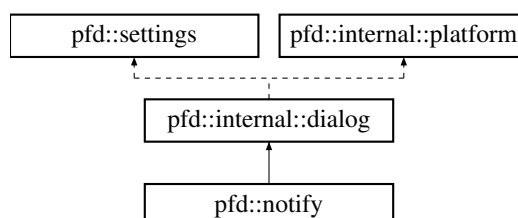
The documentation for this class was generated from the following files:

- [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](#).

```
01490 {
01491     if (_icon == icon::question) // Not supported by notifications
01492         _icon = icon::info;
01493
01494 #if _WIN32
01495     // Use a static shared pointer for notify_icon so that we can delete
01496     // it whenever we need to display a new one, and we can also wait
01497     // until the program has finished running.
01498     struct notify_icon_data : public NOTIFYICONDATAW
01499     {
01500         ~notify_icon_data() { Shell_NotifyIconW(NIM_DELETE, this); }
01501     };
01502
01503     static std::shared_ptr<notify_icon_data> nid;
01504
01505     // Release the previous notification icon, if any, and allocate a new
01506     // one. Note that std::make_shared() does value initialization, so there
01507     // is no need to memset the structure.
01508     nid = nullptr;
01509     nid = std::make_shared<notify_icon_data>();
01510
01511     // For XP support
01512     nid->cbSize = NOTIFYICONDATAW_V2_SIZE;
01513     nid->hWnd = nullptr;
01514     nid->uID = 0;
01515
01516     // Flag Description:
01517     // - NIF_ICON      The hIcon member is valid.
01518     // - NIF_MESSAGE   The uCallbackMessage member is valid.
01519     // - NIF_TIP       The szTip member is valid.
01520     // - NIF_STATE     The dwState and dwStateMask members are valid.
01521     // - NIF_INFO      Use a balloon ToolTip instead of a standard ToolTip. The szInfo, uTimeout,
01522     // - NIF_GUID       szInfoTitle, and dwInfoFlags members are valid.
01523     // - NIF_GUID       Reserved.
01524     nid->uFlags = NIF_MESSAGE | NIF_ICON | NIF_INFO;
01525
01526     // Flag Description
01527     // - NIIF_ERROR     An error icon.
01528     // - NIIF_INFO      An information icon.
01529     // - NIIF_NONE      No icon.
01530     // - NIIF_WARNING   A warning icon.
01531     // - NIIF_ICON_MASK Version 6.0. Reserved.
01532     // - NIIF_NOSOUND   Version 6.0. Do not play the associated sound. Applies only to balloon
01533     ToolTips
01534     switch (_icon)
01535     {
01536         case icon::warning: nid->dwInfoFlags = NIIF_WARNING; break;
01537         case icon::error:   nid->dwInfoFlags = NIIF_ERROR;   break;
01538         case icon::info:    nid->dwInfoFlags = NIIF_INFO;    break;
01539         /* case icon::info: */ default: nid->dwInfoFlags = NIIF_INFO; break;
01540     }
01541
01542     ENUMRESNAMEPROC icon_enum_callback = [] (HMODULE, LPCTSTR, LPTSTR lpName, LONG_PTR lParam) -> BOOL
01543     {
01544         ((NOTIFYICONDATA *)lParam)->hIcon = ::LoadIcon(GetModuleHandle(nullptr), lpName);
01545         return false;
01546     };
01547 }
```

```

01545     nid->hIcon = ::LoadIcon(nullptr, IDI_APPLICATION);
01546     ::EnumResourceNames(nullptr, RT_GROUP_ICON, icon_enum_callback, (LONG_PTR)nid.get());
01547
01548     nid->uTimeout = 5000;
01549
01550     StringCchCopyW(nid->szInfoTitle, ARRAYSIZE(nid->szInfoTitle), internal::str2wstr(title).c_str());
01551     StringCchCopyW(nid->szInfo, ARRAYSIZE(nid->szInfo), internal::str2wstr(message).c_str());
01552
01553     // Display the new icon
01554     Shell_NotifyIconW(NIM_ADD, nid.get());
01555 #elif __EMSCRIPTEN__
01556     // FIXME: do something
01557     (void)title;
01558     (void)message;
01559 #else
01560     auto command = desktop_helper();
01561
01562     if (is_osascript())
01563     {
01564         command.push_back("-e");
01565         command.push_back("display notification " + osascript_quote(message) +
01566             " with title " + osascript_quote(title));
01567     }
01568     else if (is_zenity())
01569     {
01570         command.push_back("--notification");
01571         command.push_back("--window-icon");
01572         command.push_back(get_icon_name(_icon));
01573         command.push_back("--text");
01574         command.push_back(title + "\n" + message);
01575     }
01576     else if (is_kdialog())
01577     {
01578         command.push_back("--icon");
01579         command.push_back(get_icon_name(_icon));
01580         command.push_back("--title");
01581         command.push_back(title);
01582         command.push_back("--passivepopup");
01583         command.push_back(message);
01584         command.push_back("5");
01585     }
01586
01587     if (flags(flag::is_verbose))
01588         std::cerr << "pfd: " << command << std::endl;
01589
01590     m_async->start_process(command);
01591 #endif
01592 }

```

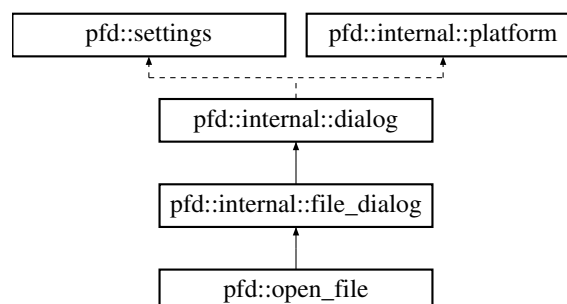
The documentation for this class was generated from the following file:

- [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](#).

```
01829         : file_dialog(type::open, title, default_path, filters, options)
01830     {
01831     }
```

**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](#).

```
01837         : open_file(title, default_path, filters,
01838             (allow_multiselect ? opt::multiselect : opt::none))
01839     {
01840     }
```

### 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     }
```

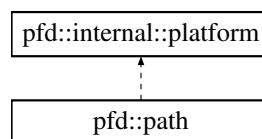
The documentation for this class was generated from the following file:

- [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](#).

```
00638 {
00639 #if _WIN32
00640     // First try the USERPROFILE environment variable
00641     auto user_profile = internal::getenv("USERPROFILE");
00642     if (user_profile.size() > 0)
00643         return user_profile;
00644     // Otherwise, try GetUserProfileDirectory()
00645     HANDLE token = nullptr;
00646     DWORD len = MAX_PATH;
00647     char buf[MAX_PATH] = { '\0' };
00648     if (OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &token))
00649     {
00650         dll userenv("userenv.dll");
00651         dll::proc<BOOL WINAPI (HANDLE, LPSTR, LPDWORD)> get_user_profile_directory(userenv,
00652 "GetUserProfileDirectoryA");
00653         get_user_profile_directory(token, buf, &len);
00654         CloseHandle(token);
00655         if (*buf)
00656             return buf;
00657     }
00658 #elif __EMSCRIPTEN__
00659     return "/";
00660 #else
00661     // First try the HOME environment variable
00662     auto home = internal::getenv("HOME");
00663     if (home.size() > 0)
00664         return home;
00665     // Otherwise, try getpwuid_r()
00666     size_t len = 4096;
00667     #if defined(_SC_GETPW_R_SIZE_MAX)
00668         auto size_max = sysconf(_SC_GETPW_R_SIZE_MAX);
00669         if (size_max != -1)
00670             len = size_t(size_max);
00671     #endif
00672     std::vector<char> buf(len);
00673     struct passwd pwd, *result;
00674     if (getpwuid_r(getuid(), &pwd, buf.data(), buf.size(), &result) == 0)
00675         return result->pw_dir;
00676     #endif
00677     return "/";
00678 }
```

## 7.28.2.2 separator()

```
std::string pfd::path::separator ( ) [inline], [static]
```

Definition at line 679 of file [FileDialog.h](#).

```
00680 {
00681 #if _WIN32
00682     return "\\";
00683 #else
00684     return "/";
00685 #endif
00686 }
```

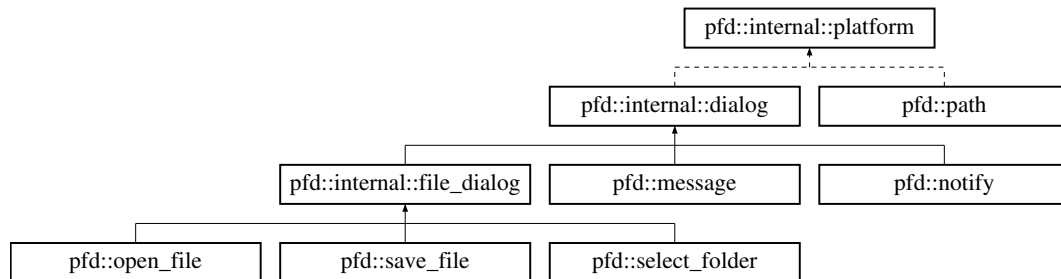
The documentation for this class was generated from the following file:

- [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](#).

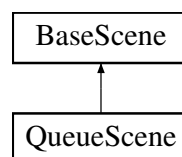
The documentation for this class was generated from the following file:

- [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](#).

```
00111 {
00112     if (this->isMenuOpen)
00113         this->menu->pollEvents(event, mousePosView);
00114
00115     this->controlMenu->pollEvents(event, mousePosView);
00116 }
```

### 7.30.3.2 popModeEvents()

```
std::vector< EventAnimation > QueueScene::popModeEvents (
    int chosenNode )
```

Definition at line 260 of file [QueueScene.cpp](#).

```
00260 {
00261     this->linkedList->resetEvents();
00262     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00263         return {};
00264
00265     this->linkedList->initHighlighter(
00266         constants::Highlighter::SLL::CODES_PATH[1].second,
00267         constants::Highlighter::SLL::CODES_PATH[1].first
00268     );
00269
00270     std::vector<EventAnimation> events;
00271     EventAnimation event;
00272
00273     if (!chosenNode) {
00274         event.titleNodes.emplace_back(chosenNode, "head|temp");
00275         event.colorNodes.push_back(chosenNode);
00276         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00277         event.lines = {0, 1};
00278
00279         events.emplace_back(event);
00280
00281         if (this->linkedList->getSize() > 1) {
00282             event.reset();
00283             event.titleNodes = {
00284                 {chosenNode, "temp"},
00285                 {1, "head"}
00286             };
00287             event.colorNodes.push_back(1);
00288             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00289             event.isPrintNormal = true;
00290             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00291             event.lines = {2};
00292
00293             events.emplace_back(event);
00294         }
00295
00296         event.reset();
00297         event.titleNodes.emplace_back(1, "head");
00298         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00299         event.lines = {3};
00300
00301         events.emplace_back(event);
00302     } else {
00303         event.reset();
00304         event.titleNodes.emplace_back(0, "head|current");
00305         event.colorNodes.push_back(0);
00306         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00307         event.lines = {5};
00308
00309         events.emplace_back(event);
00310     }
```

```

00311         for (int i = 0; i < chosenNode; ++i) {
00312             event.reset();
00313             event.titleNodes = {
00314                 {0, "head"},
00315                 {i, "current"}
00316             };
00317             event.colorNodes.push_back(i);
00318             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00319             event.lines = {6};
00320
00321             events.emplace_back(event);
00322
00323             if (i == chosenNode - 1) break;
00324
00325             event.reset();
00326             event.titleNodes = {
00327                 {0, "head"},
00328                 {i, "current"}
00329             };
00330             event.colorNodes.push_back(i);
00331             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00332             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00333             event.lines = {7};
00334
00335             events.emplace_back(event);
00336         }
00337
00338         event.reset();
00339         event.titleNodes = {
00340             {0, "head"},
00341             {chosenNode, "temp"},
00342             {chosenNode - 1, "current"}
00343         };
00344         event.colorNodes.push_back(chosenNode);
00345         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00346         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00347         event.lines = {8};
00348
00349         events.emplace_back(event);
00350
00351         if (chosenNode != this->linkedList->getSize() - 1) {
00352             event.reset();
00353             event.titleNodes = {
00354                 {0, "head"},
00355                 {chosenNode, "temp"},
00356                 {chosenNode - 1, "current"}
00357             };
00358             event.colorNodes.push_back(chosenNode);
00359             event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00360             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00361             event.isPrintNormal = true;
00362             event.lines = {9};
00363
00364             events.emplace_back(event);
00365
00366             event.reset();
00367             event.titleNodes.emplace_back(0, "head");
00368             event.statusChosenNode = NodeInfo::StatusNode::Visible;
00369             event.lines = {10};
00370
00371             events.emplace_back(event);
00372         } else {
00373             event.reset();
00374             event.titleNodes = {
00375                 {0, "head"},
00376                 {chosenNode, "temp"},
00377                 {chosenNode - 1, "current"}
00378             };
00379             event.colorNodes.push_back(chosenNode);
00380             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00381             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00382             event.lines = {9};
00383
00384             events.emplace_back(event);
00385
00386             event.reset();
00387             event.titleNodes.emplace_back(0, "head");
00388             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00389             event.statusChosenNode = NodeInfo::StatusNode::Visible;
00390             event.lines = {10};
00391
00392             events.emplace_back(event);
00393         }
00394     }
00395
00396     return events;
00397 }

```

### 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"},
```

```

00200         {i, "current"}
00201     };
00202     event.colorNodes.push_back(i);
00203     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00204     if (chosenNode == this->linkedList->getSize())
00205         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00206     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00207     event.lines = {6};
00208
00209     events.emplace_back(event);
00210
00211     if (i == chosenNode - 1) break;
00212
00213     event.reset();
00214     event.titleNodes = {
00215         {0, "head"},
00216         {chosenNode, "temp"},
00217         {i, "current"}
00218     };
00219     event.colorNodes.push_back(i);
00220     event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00221     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00222     if (chosenNode == this->linkedList->getSize())
00223         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00224     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00225     event.lines = {7};
00226
00227     events.emplace_back(event);
00228 }
00229
00230 if (chosenNode != this->linkedList->getSize()) {
00231     event.reset();
00232     event.titleNodes = {
00233         {0, "head"},
00234         {chosenNode, "temp"},
00235         {chosenNode - 1, "current"}
00236     };
00237     event.colorNodes.push_back(chosenNode);
00238     event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00239     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00240     event.isPrintNormal = true;
00241     event.lines = {8};
00242
00243     events.emplace_back(event);
00244 }
00245
00246 event.reset();
00247 event.titleNodes = {
00248     {0, "head"},
00249     {chosenNode, "temp"}
00250 };
00251 event.statusChosenNode = NodeInfo::StatusNode::InChain;
00252 event.lines = {9};
00253
00254 events.emplace_back(event);
00255 }
00256
00257 return events;
00258 }

```

### 7.30.3.4 render()

void QueueScene::render ( ) [override], [virtual]

Implements [BaseScene](#).

Definition at line 100 of file [QueueScene.cpp](#).

```

00100     {
00101         if (this->isMenuOpen)
00102             this->menu->render();
00103
00104         if (this->isDemoCodeOpen)
00105             this->linkedList->renderHighlighter();
00106
00107         this->controlMenu->render();
00108         this->linkedList->render();
00109     }

```

### 7.30.3.5 reset()

```
void QueueScene::reset ( )
```

Definition at line 123 of file [QueueScene.cpp](#).

```
00123     {
00124         this->menu->resetActiveOptionsMenu();
00125     }
```

### 7.30.3.6 update()

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

Implements [BaseScene](#).

Definition at line 11 of file [QueueScene.cpp](#).

```
00011     {
00012         if (this->isMenuOpen) {
00013             this->menu->update();
00014
00015             constants::MenuDataStructure::Button status = this->menu->getActiveOptionsMenu();
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 ==
00028 constants::MenuDataStructure::CreateMode::Button::DEFINED_LIST_BUTTON) {
00029                         if (this->menu->createModeValue[1] == "None")
00030                             break;
00031                         std::vector<std::string> values;
00032                         std::string value = this->menu->createModeValue[1];
00033                         std::stringstream ss(value);
00034                         std::string token;
00035                         while (std::getline(ss, token, ',')) {
00036                             values.push_back(token);
00037                         }
00038                         this->linkedList->createLinkedList(values);
00039                     } else if (createMode ==
00040 constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON) {
00041                         if (this->menu->createModeValue[2] == "None")
00042                             break;
00043                         std::vector<std::string> values;
00044                         std::string value = this->menu->createModeValue[2];
00045                         std::stringstream ss(value);
00046                         std::string token;
00047                         while (std::getline(ss, token, ','))
00048                             values.push_back(token);
00049                         this->linkedList->createLinkedList(values);
00050                         this->menu->createModeValue[2] = "None";
00051                     }
00052                     this->controlMenu->reset();
00053                     break;
00054                 case constants::MenuDataStructure::Button::PUSH_BUTTON:
00055                     if (this->menu->pushModeValue == "None")
00056                         break;
00057                     this->linkedList->addNode(
00058                         this->linkedList->getSize(),
00059                         this->menu->pushModeValue,
00060                         this->pushModeEvents(this->linkedList->getSize())
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->getActiveOptionsMenu() !=
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->resetActiveOptionsMenuOnly();
00077         this->controlMenu->reset();
00078         break;
00079     case constants::MenuDataStructure::Button::CLEAR_BUTTON:
00080         if (this->menu->getActiveOptionsMenu() !=
constants::MenuDataStructure::Button::CLEAR_BUTTON)
00081             break;
00082
00083         this->linkedList->createLinkedList(0);
00084
00085         std::cout << "Cleared " << std::endl;
00086         this->menu->resetActiveOptionsMenuOnly();
00087         this->controlMenu->reset();
00088         break;
00089     }
00090 }
00091
00092 this->controlMenu->update();
00093
00094 this->linkedList->processControlMenu(this->controlMenu->getStatus());
00095 this->linkedList->setSpeed(this->controlMenu->getSpeed());
00096
00097 this->linkedList->update();
00098 }

```

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

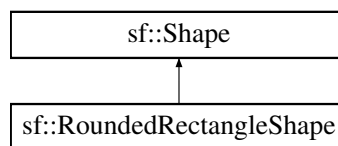
- include/libScene/QueueScene.hpp
- include/libScene/QueueScene.cpp

## 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 cornerPointCount=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

<i>size</i>	Size of the rectangle
<i>radius</i>	Radius for each rounded corner
<i>cornerPointCount</i>	Number of points of each corner



Definition at line 31 of file [RoundedRectangleShape.cpp](#).

```
00032     {
00033         mySize = size;
00034         myRadius = radius;
00035         myCornerPointCount = cornerPointCount;
00036         update();
00037     }
```

### 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](#).

```
00061     {
00062         return myRadius;
00063     }
```

#### 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

<i>index</i>	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](#).

```

00080     {
00081         if(index >= myCornerPointCount*4)
00082             return sf::Vector2f(0,0);
00083
00084         float deltaAngle = 90.0f/(myCornerPointCount-1);
00085         sf::Vector2f center;
00086         unsigned int centerIndex = index/myCornerPointCount;
00087         static const float pi = 3.141592654f;
00088
00089         switch(centerIndex)
00090         {
00091             case 0: center.x = mySize.x - myRadius; center.y = myRadius; break;
00092             case 1: center.x = myRadius; center.y = myRadius; break;
00093             case 2: center.x = myRadius; center.y = mySize.y - myRadius; break;
00094             case 3: center.x = mySize.x - myRadius; center.y = mySize.y - myRadius; break;
00095         }
00096
00097         return sf::Vector2f(myRadius*cos(deltaAngle*(index-centerIndex)*pi/180)+center.x,
00098                             -myRadius*sin(deltaAngle*(index-centerIndex)*pi/180)+center.y);
00099     }

```

### 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](#).

```

00074     {
00075         return myCornerPointCount*4;
00076     }

```

### 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](#).

```

00048     {
00049         return mySize;
00050     }

```

### 7.31.3.5 setCornerPointCount()

```
void sf::RoundedRectangleShape::setCornerPointCount (
    unsigned int count )
```

Set the number of points of each corner.

## Parameters

<i>count</i>	New number of points of the rounded rectangle
--------------	---

## See also

[getPointCount](#)

Definition at line 66 of file [RoundedRectangleShape.cpp](#).

```
00067     {  
00068         myCornerPointCount = count;  
00069         update();  
00070     }
```

### 7.31.3.6 setCornersRadius()

```
void sf::RoundedRectangleShape::setCornersRadius (  
    float radius )
```

Set the radius of the rounded corners.

## Parameters

<i>radius</i>	Radius of the rounded corners
---------------	-------------------------------

## See also

[getCornersRadius](#)

Definition at line 53 of file [RoundedRectangleShape.cpp](#).

```
00054     {  
00055         myRadius = radius;  
00056         update();  
00057     }
```

### 7.31.3.7 setSize()

```
void sf::RoundedRectangleShape::setSize (  
    const Vector2f & size )
```

Set the size of the rounded rectangle.

## Parameters

<i>size</i>	New size of the rounded rectangle
-------------	-----------------------------------

See also

[getSize](#)

Definition at line 40 of file [RoundedRectangleShape.cpp](#).

```
00041     {
00042         mySize = size;
00043         update();
00044     }
```

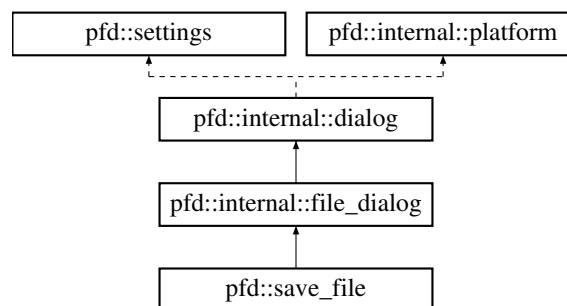
The documentation for this class was generated from the following files:

- [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](#).

```
01853         : file_dialog(type::save, title, default_path, filters, options)
01854     {
01855     }
```

### 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](#).

```
01861         : save_file(title, default_path, filters,
01862                     (confirm_overwrite ? opt::none : opt::force_overwrite))
01863     {
01864     }
```

## 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](#).

```
01867     {
01868         return string_result();
01869     }
```

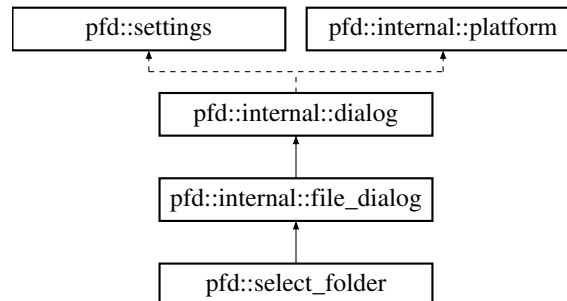
The documentation for this class was generated from the following file:

- [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](#).

```
01881 {
01882     return string_result();
01883 }
```

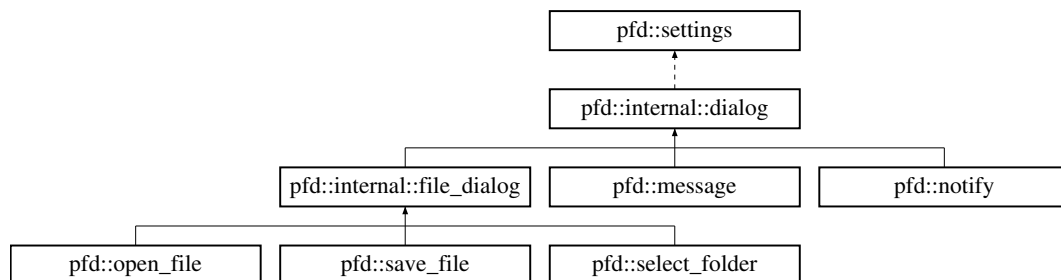
The documentation for this class was generated from the following file:

- [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](#).

```
00139     {
00140         is_scanned = 0,
00141         is_verbose,
00142
00143         has_zenity,
00144         has_matedialog,
00145         has_qarma,
00146         has_kdialog,
00147         is_vista,
00148
00149         max_flag,
00150     };
```

### 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](#).

```
00525     {
00526         flags(flag::is_scanned) &= !resync;
00527
00528         if (flags(flag::is_scanned))
00529             return;
00530
00531         auto pfd_verbose = internal::getenv("PFD_VERBOSE");
00532         auto match_no = std::regex("(0|no|false)", std::regex_constants::icase);
00533         if (!std::regex_match(pfd_verbose, match_no))
00534             flags(flag::is_verbose) = true;
00535
00536         #if _WIN32
00537             flags(flag::is_vista) = internal::is_vista();
00538         #elif !__APPLE__
00539             flags(flag::has_zenity) = check_program("zenity");
00540             flags(flag::has_matedialog) = check_program("matedialog");
00541             flags(flag::has_qarma) = check_program("qarma");
00542             flags(flag::has_kdialog) = check_program("kdialog");
00543
00544             // If multiple helpers are available, try to default to the best one
00545             if (flags(flag::has_zenity) && flags(flag::has_kdialog))
00546             {
00547                 auto desktop_name = internal::getenv("XDG_SESSION_DESKTOP");
00548                 if (desktop_name == std::string("gnome"))
00549                     flags(flag::has_kdialog) = false;
00550                 else if (desktop_name == std::string("KDE"))
00551                     flags(flag::has_zenity) = false;
00552             }
00553         #endif
00554
00555         flags(flag::is_scanned) = true;
00556     }
```

## 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](#).

```
00559     {
00560         #if _WIN32
00561             return true;
00562         #elif __APPLE__
00563             return true;
00564         #elif __EMSCRIPTEN__
00565             // FIXME: Return true after implementation is complete.
00566             return false;
00567         #else
00568             settings tmp;
00569             return tmp.flags(flag::has_zenity) ||
00570                    tmp.flags(flag::has_matedialog) ||
00571                    tmp.flags(flag::has_qarma) ||
00572                    tmp.flags(flag::has_kdialog);
00573         #endif
00574     }
```

#### 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](#).

```
00588     {
00589     #if _WIN32
00590         (void)program;
00591         return false;
00592     #elif __EMSCRIPTEN__
00593         (void)program;
00594         return false;
00595     #else
00596         int exit_code = -1;
00597         internal::executor async;
00598         async.start_process({"/bin/sh", "-c", "which " + program});
00599         async.result(&exit_code);
00600         return exit_code == 0;
00601     #endif
00602     }
```

#### 7.34.4.3 flags() [1/2]

```
bool & pfd::settings::flags (
    flag in_flag ) [inline], [protected]
```

Definition at line 631 of file [FileDialog.h](#).

```
00632     {
00633         return const_cast<bool &>(static_cast<settings const *>(this)->flags(in_flag));
00634     }
```

#### 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](#).

```
00626     {
00627         static bool flags[size_t(flag::max_flag)];
00628         return flags[size_t(in_flag)];
00629     }
```

#### 7.34.4.5 is\_kdialog()

```
bool pfd::settings::is_kdialog ( ) const [inline], [protected]
```

Definition at line 620 of file [FileDialog.h](#).

```
00621     {
00622         return flags(flag::has_kdialog);
00623     }
```

#### 7.34.4.6 is\_osascript()

```
bool pfd::settings::is_osascript ( ) const [inline], [protected]
```

Definition at line 604 of file [FileDialog.h](#).

```
00605 {
00606 #if __APPLE__
00607     return true;
00608 #else
00609     return false;
00610 #endif
00611 }
```

#### 7.34.4.7 is\_zenity()

```
bool pfd::settings::is_zenity ( ) const [inline], [protected]
```

Definition at line 613 of file [FileDialog.h](#).

```
00614 {
00615     return flags(flag::has_zenity) ||
00616            flags(flag::has_matedialog) ||
00617            flags(flag::has_qarma);
00618 }
```

#### 7.34.4.8 rescan()

```
void pfd::settings::rescan ( ) [inline], [static]
```

Definition at line 581 of file [FileDialog.h](#).

```
00582 {
00583     settings(/* resync = */ true);
00584 }
```

#### 7.34.4.9 verbose()

```
void pfd::settings::verbose (
    bool value ) [inline], [static]
```

Definition at line 576 of file [FileDialog.h](#).

```
00577 {
00578     settings().flags(flag::is_verbose) = value;
00579 }
```

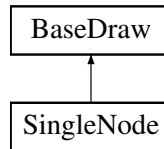
The documentation for this class was generated from the following file:

- [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](#).

```
00007                                     : BaseDraw(window)
{
00008     this->value = std::move(value);
00009
00010     this->circle.setRadius(constants::NodeInfo::radius);
00011     this->circle.setFillColor(sf::Color::White);
00012     this->circle.setOutlineThickness(constants::NodeInfo::outlineThickness);
00013     this->circle.setOutlineColor(sf::Color::Black);
00014     this->circle.setPointCount(constants::NodeInfo::pointCount);
00015     sf::FloatRect bounds = this->circle.getLocalBounds();
00016     this->circle.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00017     this->circle.setPosition(position);
00018
00019     this->font.loadFromFile(constants::FontPath);
00020     this->label.setFont(this->font);
00021     this->label.setString(this->value);
00022     this->label.setCharacterSize(constants::NodeInfo::fontSize);
00023     this->label.setFillColor(sf::Color::Black);
00024     bounds = this->label.getLocalBounds();
00025     this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00026     this->label.setPosition(position);
00027 }
```

## 7.35.3 Member Function Documentation

### 7.35.3.1 getPosition()

```
sf::Vector2f SingleNode::getPosition ( )
```

Definition at line 47 of file [SingleNode.cpp](#).

```
00047     {
00048         return this->circle.getPosition();
00049     }
```

### 7.35.3.2 render()

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

Implements [BaseDraw](#).

Definition at line 29 of file [SingleNode.cpp](#).

```
00029     {
00030         this->window->draw(this->circle);
00031         this->window->draw(this->label);
00032     }
```

### 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](#).

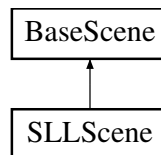
The documentation for this class was generated from the following files:

- [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](#).

```
00130                                     : BaseScene(window) {
00131     this->init();
00132 }
```

## 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](#).

```
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 }
```

```

00161     else {
00162         event.titleNodes.emplace_back(chosenNode, "temp");
00163         if (this->linkedList->getSize())
00164             event.titleNodes.emplace_back(1, "head");
00165     }
00166     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00167     if (chosenNode && chosenNode == this->linkedList->getSize())
00168         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00169     event.colorNodes.push_back(chosenNode);
00170     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00171     event.lines = {0};
00172
00173     events.emplace_back(event);
00174
00175     if (chosenNode == 0) {
00176         if (this->linkedList->getSize()) {
00177             event.reset();
00178             event.titleNodes = {
00179                 {1, "head"},
00180                 {chosenNode, "temp"}
00181             };
00182             event.colorNodes = std::vector<int>{0};
00183             event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
00184             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00185             event.isPrintNormal = true;
00186             event.lines = {1, 2};
00187
00188             events.emplace_back(event);
00189         }
00190
00191         event.reset();
00192         event.titleNodes.emplace_back(chosenNode, "head|temp");
00193         event.lines = {3};
00194         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00195         events.emplace_back(event);
00196     } else {
00197         event.reset();
00198         event.titleNodes = {
00199             {0, "head|current"},
00200             {chosenNode, "temp"}
00201         };
00202         event.colorNodes.push_back(0);
00203         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00204         if (chosenNode == this->linkedList->getSize())
00205             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00206         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00207         event.lines = {5};
00208
00209         events.emplace_back(event);
00210
00211         for (int i = 0; i < chosenNode; ++i) {
00212             event.reset();
00213             event.titleNodes = {
00214                 {0, "head"},
00215                 {chosenNode, "temp"},
00216                 {i, "current"}
00217             };
00218             event.colorNodes.push_back(i);
00219             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00220             if (chosenNode == this->linkedList->getSize())
00221                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00222             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00223             event.lines = {6};
00224
00225             events.emplace_back(event);
00226
00227             if (i == chosenNode - 1) break;
00228
00229             event.reset();
00230             event.titleNodes = {
00231                 {0, "head"},
00232                 {chosenNode, "temp"},
00233                 {i, "current"}
00234             };
00235             event.colorNodes.push_back(i);
00236             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00237             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00238             if (chosenNode == this->linkedList->getSize())
00239                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00240             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00241             event.lines = {7};
00242
00243             events.emplace_back(event);
00244         }
00245
00246         if (chosenNode != this->linkedList->getSize()) {
00247             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();

```

```
00320     event.titleNodes.emplace_back(0, "head|current");
00321     event.colorNodes.push_back(0);
00322     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00323     event.lines = {5};
00324
00325     events.emplace_back(event);
00326
00327     for (int i = 0; i < chosenNode; ++i) {
00328         event.reset();
00329         event.titleNodes = {
00330             {0, "head"},
00331             {i, "current"}
00332         };
00333         event.colorNodes.push_back(i);
00334         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00335         event.lines = {6};
00336
00337         events.emplace_back(event);
00338
00339         if (i == chosenNode - 1) break;
00340
00341         event.reset();
00342         event.titleNodes = {
00343             {0, "head"},
00344             {i, "current"}
00345         };
00346         event.colorNodes.push_back(i);
00347         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00348         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00349         event.lines = {7};
00350
00351         events.emplace_back(event);
00352     }
00353
00354     event.reset();
00355     event.titleNodes = {
00356         {0, "head"},
00357         {chosenNode, "temp"},
00358         {chosenNode - 1, "current"}
00359     };
00360     event.colorNodes.push_back(chosenNode);
00361     event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00362     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00363     event.lines = {8};
00364
00365     events.emplace_back(event);
00366
00367     if (chosenNode != this->linkedList->getSize() - 1) {
00368         event.reset();
00369         event.titleNodes = {
00370             {0, "head"},
00371             {chosenNode, "temp"},
00372             {chosenNode - 1, "current"}
00373         };
00374         event.colorNodes.push_back(chosenNode);
00375         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00376         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00377         event.isPrintNormal = true;
00378         event.lines = {9};
00379
00380         events.emplace_back(event);
00381
00382         event.reset();
00383         event.titleNodes.emplace_back(0, "head");
00384         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00385         event.lines = {10};
00386
00387         events.emplace_back(event);
00388     } else {
00389         event.reset();
00390         event.titleNodes = {
00391             {0, "head"},
00392             {chosenNode, "temp"},
00393             {chosenNode - 1, "current"}
00394         };
00395         event.colorNodes.push_back(chosenNode);
00396         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00397         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00398         event.lines = {9};
00399
00400         events.emplace_back(event);
00401
00402         event.reset();
00403         event.titleNodes.emplace_back(0, "head");
00404         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00405         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00406         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->resetActiveOptionsMenu();
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->getActiveOptionsMenu();
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         }
00462     }

```

```

00462     }
00463
00464     event.reset();
00465     if (chosenNode == 0)
00466         event.titleNodes.emplace_back(0, "head|current");
00467     else
00468         event.titleNodes = {
00469             {0, "head"},
00470             {chosenNode, "current"}
00471         };
00472     event.lines = {3};
00473
00474     events.emplace_back(event);
00475
00476     return events;
00477 }

```

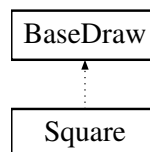
The documentation for this class was generated from the following files:

- 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](#).

```
00013     {
00014         inactive,
00015         active,
00016         chosen,
00017         hidden
00018     };
```

### 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](#).

```
00008     : BaseDraw(window) {
00009     this->value = std::move(value);
00010
00011     this->square.setSize(sf::Vector2f(constants::Square::length, constants::Square::length));
00012     this->square.setFillColor(sf::Color::White);
00013     this->square.setOutlineThickness(constants::Square::outlineThickness);
00014     this->square.setOutlineColor(sf::Color::Black);
00015     sf::FloatRect bounds = this->square.getLocalBounds();
00016     this->square.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00017     this->square.setPosition(position);
00018
00019     this->font.loadFromFile(constants::fontPath);
00020     this->label.setFont(this->font);
00021     this->label.setString(this->value);
00022     this->label.setCharacterSize(constants::Square::fontSize);
00023     this->label.setFillColor(sf::Color::Black);
00024     bounds = this->label.getLocalBounds();
00025     this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00026     this->label.setPosition(position);
00027 }
```

## 7.37.4 Member Function Documentation

### 7.37.4.1 getPosition()

```
sf::Vector2f Square::getPosition ( )
```

Definition at line 62 of file [Square.cpp](#).

```
00062         {  
00063     return this->square.getPosition();  
00064 }
```

### 7.37.4.2 getStatus()

```
Square::Status Square::getStatus ( )
```

Definition at line 70 of file [Square.cpp](#).

```
00070     {  
00071     return this->status;  
00072 }
```

### 7.37.4.3 render()

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

Implements [BaseDraw](#).

Definition at line 29 of file [Square.cpp](#).

```
00029     {  
00030     switch (this->status) {  
00031     case Status::active:  
00032         this->square.setOutlineColor(constants::normalGreen);  
00033         break;  
00034     case Status::inactive:  
00035         this->square.setOutlineColor(sf::Color::Black);  
00036         break;  
00037     case Status::chosen:  
00038         this->square.setOutlineColor(constants::clickGreen);  
00039         break;  
00040     }  
00041     this->window->draw(this->square);  
00042     this->window->draw(this->label);  
00043 }
```

### 7.37.4.4 resetColor()

```
void Square::resetColor ( )
```

Definition at line 45 of file [Square.cpp](#).

```
00045     {  
00046     this->status = Status::inactive;  
00047 }
```

#### 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](#).

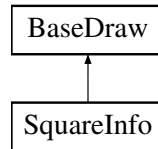
The documentation for this class was generated from the following files:

- [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](#).

```
00007                                     : BaseDraw(window)
    {
00008         this->position = position;
00009         this->square = new Square(window, value, position);
00010         this->values[0] = std::move(value);
00011         this->values[1] = "";
00012         this->isPrintPreVal = false;
00013
00014         this->title.setFont(this->square->font);
00015         this->title.setCharacterSize(20);
00016         this->title.setFillColor(sf::Color::Black);
00017     }
```

### 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](#).

```
00067     {
00068         return this->square->getStatus();
00069     }
```

### 7.38.3.2 getValue()

```
std::string SquareInfo::getValue ( )
```

Definition at line 59 of file [SquareInfo.cpp](#).

```
00059     {
00060         return this->values[0];
00061     }
```

### 7.38.3.3 render()

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

Implements [BaseDraw](#).

Definition at line 19 of file [SquareInfo.cpp](#).

```
00019 {
00020     if (this->square->getStatus() != Square::Status::hidden) {
00021         this->square->render();
00022         this->window->draw(this->title);
00023     }
00024 }
```

### 7.38.3.4 reset()

```
void SquareInfo::reset ( )
```

Definition at line 49 of file [SquareInfo.cpp](#).

```
00049 {
00050     this->resetTitle();
00051     this->square->resetColor();
00052     this->isPrintPreVal = false;
00053 }
```

### 7.38.3.5 resetTitle()

```
void SquareInfo::resetTitle ( )
```

Definition at line 45 of file [SquareInfo.cpp](#).

```
00045 {
00046     this->title.setString("");
00047 }
```

### 7.38.3.6 setPrintPreVal()

```
void SquareInfo::setPrintPreVal (
    bool _isPrintPreVal )
```

Definition at line 63 of file [SquareInfo.cpp](#).

```
00063 {
00064     this->isPrintPreVal = _isPrintPreVal;
00065 }
```



### 7.38.3.7 setStatus()

```
void SquareInfo::setStatus (
    Square::Status _status )
```

Definition at line 55 of file [SquareInfo.cpp](#).

```
00055 {
00056     this->square->setStatus(_status);
00057 }
```

### 7.38.3.8 setTitle()

```
void SquareInfo::setTitle (
    const std::string & _title )
```

Definition at line 38 of file [SquareInfo.cpp](#).

```
00038 {
00039     this->title.setString(_title);
00040     sf::FloatRect bounds = this->title.getLocalBounds();
00041     this->title.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00042     this->title.setPosition(this->position.x, this->position.y + constants::TitleNode::offsetY);
00043 }
```

### 7.38.3.9 setValue()

```
void SquareInfo::setValue (
    std::string value )
```

Definition at line 26 of file [SquareInfo.cpp](#).

```
00026 {
00027     this->values[1] = this->values[0];
00028     this->values[0] = std::move(value);
00029 }
```

### 7.38.3.10 update()

```
void SquareInfo::update ( )
```

Definition at line 31 of file [SquareInfo.cpp](#).

```
00031 {
00032     if (this->isPrintPreVal)
00033         this->square->setText(this->values[1]);
00034     else
00035         this->square->setText(this->values[0]);
00036 }
```

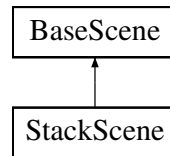
The documentation for this class was generated from the following files:

- [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](#).

```
00007                                     : BaseScene(window) {
00008     this->init();
00009 }
```

### 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](#).

```
00111                                     {
00112     if (this->isMenuOpen)
00113         this->menu->pollEvents(event, mousePosView);
00114
00115     this->controlMenu->pollEvents(event, mousePosView);
00116 }
```

### 7.39.3.2 popModeEvents()

```
std::vector< EventAnimation > StackScene::popModeEvents (
    int chosenNode )
```

Definition at line 260 of file [StackScene.cpp](#).

```
00260 {
00261     this->linkedList->resetEvents();
00262     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00263         return {};
00264
00265     this->linkedList->initHighlighter(
00266         constants::Highlighter::SLL::CODES_PATH[1].second,
00267         constants::Highlighter::SLL::CODES_PATH[1].first
00268     );
00269
00270     std::vector<EventAnimation> events;
00271     EventAnimation event;
00272
00273     if (!chosenNode) {
00274         event.titleNodes.emplace_back(chosenNode, "head|temp");
00275         event.colorNodes.push_back(chosenNode);
00276         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00277         event.lines = {0, 1};
00278
00279         events.emplace_back(event);
00280
00281         if (this->linkedList->getSize() > 1) {
00282             event.reset();
00283             event.titleNodes = {
00284                 {chosenNode, "temp"},
00285                 {1, "head"}
00286             };
00287             event.colorNodes.push_back(1);
00288             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00289             event.isPrintNormal = true;
00290             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00291             event.lines = {2};
00292
00293             events.emplace_back(event);
00294         }
00295
00296         event.reset();
00297         event.titleNodes.emplace_back(1, "head");
00298         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00299         event.lines = {3};
00300
00301         events.emplace_back(event);
00302     } else {
00303         event.reset();
00304         event.titleNodes.emplace_back(0, "head|current");
00305         event.colorNodes.push_back(0);
00306         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00307         event.lines = {5};
00308
00309         events.emplace_back(event);
00310
00311         for (int i = 0; i < chosenNode; ++i) {
00312             event.reset();
00313             event.titleNodes = {
00314                 {0, "head"},
00315                 {i, "current"}
00316             };
00317             event.colorNodes.push_back(i);
00318             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00319             event.lines = {6};
00320
00321             events.emplace_back(event);
00322
00323             if (i == chosenNode - 1) break;
00324
00325             event.reset();
00326             event.titleNodes = {
00327                 {0, "head"},
00328                 {i, "current"}
00329             };
00330             event.colorNodes.push_back(i);
00331             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00332             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00333             event.lines = {7};
00334
00335             events.emplace_back(event);
00336         }
00337     }
```

```

00338         event.reset();
00339         event.titleNodes = {
00340             {0, "head"},
00341             {chosenNode, "temp"},
00342             {chosenNode - 1, "current"}
00343         };
00344         event.colorNodes.push_back(chosenNode);
00345         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00346         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00347         event.lines = {8};
00348
00349         events.emplace_back(event);
00350
00351         if (chosenNode != this->linkedList->getSize() - 1) {
00352             event.reset();
00353             event.titleNodes = {
00354                 {0, "head"},
00355                 {chosenNode, "temp"},
00356                 {chosenNode - 1, "current"}
00357             };
00358             event.colorNodes.push_back(chosenNode);
00359             event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00360             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00361             event.isPrintNormal = true;
00362             event.lines = {9};
00363
00364             events.emplace_back(event);
00365
00366             event.reset();
00367             event.titleNodes.emplace_back(0, "head");
00368             event.statusChosenNode = NodeInfo::StatusNode::Visible;
00369             event.lines = {10};
00370
00371             events.emplace_back(event);
00372         } else {
00373             event.reset();
00374             event.titleNodes = {
00375                 {0, "head"},
00376                 {chosenNode, "temp"},
00377                 {chosenNode - 1, "current"}
00378             };
00379             event.colorNodes.push_back(chosenNode);
00380             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00381             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00382             event.lines = {9};
00383
00384             events.emplace_back(event);
00385
00386             event.reset();
00387             event.titleNodes.emplace_back(0, "head");
00388             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00389             event.statusChosenNode = NodeInfo::StatusNode::Visible;
00390             event.lines = {10};
00391
00392             events.emplace_back(event);
00393         }
00394     }
00395
00396     return events;
00397 }

```

### 7.39.3.3 pushModeEvents()

```

std::vector< EventAnimation > StackScene::pushModeEvents (
    int chosenNode )

```

Definition at line 127 of file [StackScene.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"},
00200                 {i, "current"}
00201             };
00202             event.colorNodes.push_back(i);
00203             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00204             if (chosenNode == this->linkedList->getSize())
00205                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00206             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00207             event.lines = {6};
00208
00209             events.emplace_back(event);
00210
00211             if (i == chosenNode - 1) break;
00212
00213             event.reset();
00214             event.titleNodes = {
00215                 {0, "head"},
00216                 {chosenNode, "temp"},
00217                 {i, "current"}
00218             };
00219             event.colorNodes.push_back(i);
00220             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00221             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00222             if (chosenNode == this->linkedList->getSize())
00223                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00224             event.statusChosenNode = NodeInfo::StatusNode::OutChain;

```

```

00225         event.lines = {7};
00226
00227         events.emplace_back(event);
00228     }
00229
00230     if (chosenNode != this->linkedList->getSize()) {
00231         event.reset();
00232         event.titleNodes = {
00233             {0, "head"},
00234             {chosenNode, "temp"},
00235             {chosenNode - 1, "current"}
00236         };
00237         event.colorNodes.push_back(chosenNode);
00238         event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00239         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00240         event.isPrintNormal = true;
00241         event.lines = {8};
00242
00243         events.emplace_back(event);
00244     }
00245
00246     event.reset();
00247     event.titleNodes = {
00248         {0, "head"},
00249         {chosenNode, "temp"}
00250     };
00251     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00252     event.lines = {9};
00253
00254     events.emplace_back(event);
00255 }
00256
00257 return events;
00258 }

```

### 7.39.3.4 render()

void StackScene::render ( ) [override], [virtual]

Implements [BaseScene](#).

Definition at line 100 of file [StackScene.cpp](#).

```

00100     {
00101         if (this->isMenuOpen)
00102             this->menu->render();
00103
00104         if (this->isDemoCodeOpen)
00105             this->linkedList->renderHighlighter();
00106
00107         this->controlMenu->render();
00108         this->linkedList->render();
00109     }

```

### 7.39.3.5 reset()

void StackScene::reset ( )

Definition at line 123 of file [StackScene.cpp](#).

```

00123     {
00124         this->menu->resetActiveOptionsMenu();
00125     }

```

### 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->getActiveOptionsMenu();
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 ==
00028 constants::MenuDataStructure::CreateMode::Button::DEFINED_LIST_BUTTON) {
00029                         if (this->menu->createModeValue[1] == "None")
00030                             break;
00031                         std::vector<std::string> values;
00032                         std::string value = this->menu->createModeValue[1];
00033                         std::stringstream ss(value);
00034                         std::string token;
00035                         while (std::getline(ss, token, ',')) {
00036                             values.push_back(token);
00037                         }
00038                         this->linkedList->createLinkedList(values);
00039                     } else if (createMode ==
00040 constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON) {
00041                         if (this->menu->createModeValue[2] == "None")
00042                             break;
00043                         std::vector<std::string> values;
00044                         std::string value = this->menu->createModeValue[2];
00045                         std::stringstream ss(value);
00046                         std::string token;
00047                         while (std::getline(ss, token, ','))
00048                             values.push_back(token);
00049                         this->linkedList->createLinkedList(values);
00050                         this->menu->createModeValue[2] = "None";
00051                     }
00052                     this->controlMenu->reset();
00053                     break;
00054                 case constants::MenuDataStructure::Button::PUSH_BUTTON:
00055                     if (this->menu->pushModeValue == "None")
00056                         break;
00057                     this->linkedList->addNode(
00058                         0,
00059                         this->menu->pushModeValue,
00060                         this->pushModeEvents(0)
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->getActiveOptionsMenu() !=
00068 constants::MenuDataStructure::Button::POP_BUTTON)
00069                         break;
00070                     this->linkedList->deleteNode(
00071                         0,
00072                         this->popModeEvents(0)
00073                     );
00074                     std::cout << "Popped " << std::endl;
00075                     this->menu->resetActiveOptionsMenuOnly();
00076                     this->controlMenu->reset();
00077                     break;
00078                 case constants::MenuDataStructure::Button::CLEAR_BUTTON:
00079                     if (this->menu->getActiveOptionsMenu() !=
00080 constants::MenuDataStructure::Button::CLEAR_BUTTON)
00081                         break;
00082
```



```

00083         this->linkedList->createLinkedList (0);
00084
00085         std::cout << "Cleared " << std::endl;
00086         this->menu->resetActiveOptionsMenuOnly ();
00087         this->controlMenu->reset ();
00088         break;
00089     }
00090 }
00091
00092 this->controlMenu->update ();
00093
00094 this->linkedList->processControlMenu (this->controlMenu->getStatus ());
00095 this->linkedList->setSpeed (this->controlMenu->getSpeed ());
00096
00097 this->linkedList->update ();
00098 }

```

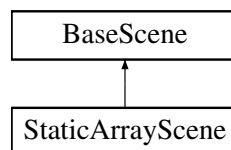
The documentation for this class was generated from the following files:

- 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](#).

```
00007                                     : BaseScene(window) {  
00008     this->init();  
00009 }
```

## 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](#).

```
00143
00144     this->array->resetEvents();
00145     if (chosenNode < 0 || chosenNode > this->array->getSize())
00146         return {};
00147
00148     // init highlighter
00149     // ...
00150
00151     int size = this->array->getSize() + 1,
00152         squaresSize = this->array->getSquaresSize();
00153     if (size > squaresSize) --size;
00154     if (!squaresSize) return {};
00155
00156     std::vector<EventAnimation> events;
00157     EventAnimation event;
00158
00159     if (size < squaresSize) {
00160         event = EventAnimation();
00161         event.eventSquares.assign(squaresSize, EventSquare());
00162         for (auto &square: event.eventSquares) {
00163             square.status = Square::Status::active;
00164             square.isPrintPreVal = true;
00165         }
00166         for (int i = size - 1; i < squaresSize; ++i)
00167             event.eventSquares[i].status = Square::Status::inactive;
00168         if (size > 1)
00169             event.eventSquares[size - 2].title = "n";
00170
00171         events.emplace_back(event);
00172
00173         event = EventAnimation();
00174         event.eventSquares.assign(squaresSize, EventSquare());
00175         for (auto &square : event.eventSquares) {
00176             square.status = Square::Status::active;
00177             square.isPrintPreVal = true;
00178         }
00179         for (int i = size; i < squaresSize; ++i)
00180             event.eventSquares[i].status = Square::Status::inactive;
00181         event.eventSquares[size - 1].title = "n";
00182
00183         events.emplace_back(event);
00184     }
00185
00186     for (int i = size - 1; i >= chosenNode; --i) {
00187         event = EventAnimation();
00188         event.eventSquares.assign(squaresSize, EventSquare());
00189         for (auto &square: event.eventSquares) {
00190             square.status = Square::Status::active;
00191             square.isPrintPreVal = true;
00192         }
00193         for (int j = size; j < squaresSize; ++j)
00194             event.eventSquares[j].status = Square::Status::inactive;
00195         event.eventSquares[size - 1].title = "n";
00196         for (int j = size - 1; j > i; --j)
00197             event.eventSquares[j].isPrintPreVal = false;
00198         event.eventSquares[i].status = Square::Status::chosen;
00199
00200         events.emplace_back(event);
00201
00202         event.eventSquares[i].isPrintPreVal = false;
00203         if (i > chosenNode)
00204             event.eventSquares[i - 1].status = Square::Status::chosen;
00205
00206         events.emplace_back(event);
00207     }
00208
00209     return events;
00210 }
```

## 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](#).

```
00116     {
00117         if (this->isMenuOpen)
00118             this->menu->render();
00119
00120         if (this->isDemoCodeOpen)
00121             this->array->renderHighlighter();
00122
00123         this->controlMenu->render();
00124         this->array->render();
00125     }
```

### 7.40.3.5 reset()

```
void StaticArrayScene::reset ( )
```

Definition at line 139 of file [StaticArrayScene.cpp](#).

```
00139     {
00140         this->menu->resetActiveOptionsMenu();
00141     }
```

### 7.40.3.6 searchModeEvents()

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

Definition at line 295 of file [StaticArrayScene.cpp](#).

```
00295     {
00296         this->array->resetEvents();
00297
00298         // init highlighter
00299         // ...
00300
00301         int size = this->array->getSize(),
00302             squaresSize = this->array->getSquaresSize();
00303         std::vector<EventAnimation> events;
00304         EventAnimation event;
00305
00306         for (int i = 0; i <= chosenNode; ++i) {
00307             if (i == size) break;
00308
00309             event = EventAnimation();
00310             event.eventSquares.assign(squaresSize, EventSquare());
00311             for (int j = 0; j < size; ++j) {
00312                 event.eventSquares[j].status = Square::Status::active;
00313                 if (j == size - 1)
00314                     event.eventSquares[size - 1].title = "n";
00315             }
00316             event.eventSquares[i].status = Square::Status::chosen;
00317
00318             events.emplace_back(event);
00319         }
00320
00321         if (chosenNode == size) {
00322             event = EventAnimation();
00323             event.eventSquares.assign(squaresSize, EventSquare());
00324             for (int j = 0; j < size; ++j) {
00325                 event.eventSquares[j].status = Square::Status::active;
00326                 if (j == size - 1)
00327                     event.eventSquares[size - 1].title = "n";
00328             }
00329
00330             events.emplace_back(event);
00331         }
00332
00333         return events;
00334     }
```

### 7.40.3.7 update()

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

Implements [BaseScene](#).

Definition at line 11 of file [StaticArrayScene.cpp](#).

```
00011     {
00012         if (this->isMenuOpen) {
00013             this->menu->update();
00014
00015             constants::MenuArray::Button status = this->menu->getActiveOptionsMenu();
00016             constants::MenuArray::CreateMode::Button createMode;
00017             switch (status){
00018                 case constants::MenuArray::Button::CREATE_BUTTON:
00019                     createMode = this->menu->getActiveCreateMode();
00020                     if (createMode == constants::MenuArray::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->array->createArray(size);
00027                     } else if (createMode ==
00028 constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
00029                         if (this->menu->createModeValue[1] == "None")
00030                             break;
00031                         std::vector<std::string> values;
00032                         std::string value = this->menu->createModeValue[1];
00033                         std::stringstream ss(value);
00034                         std::string token;
00035                         while (std::getline(ss, token, ',')) {
00036                             values.push_back(token);
00037                         }
00038                         this->array->createArray(values);
00039                     } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
00040                         if (this->menu->createModeValue[2] == "None")
00041                             break;
00042                         std::vector<std::string> values;
00043                         std::string value = this->menu->createModeValue[2];
00044                         std::stringstream ss(value);
00045                         std::string token;
00046                         while (std::getline(ss, token, ','))
00047                             values.push_back(token);
00048                         this->array->createArray(values);
00049                         this->menu->createModeValue[2] = "None";
00050                     }
00051                     this->controlMenu->reset();
00052                     break;
00053                 case constants::MenuArray::Button::ADD_BUTTON:
00054                     if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
00055 this->menu->addModeValue[0].empty())
00056                         break;
00057                     this->array->addSquare(
00058                         std::stoi(this->menu->addModeValue[0]),
00059                         this->menu->addModeValue[1],
00060                         this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00061                     );
00062                     std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
00063 << std::endl;
00064                     this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00065                     this->controlMenu->reset();
00066                     break;
00067                 case constants::MenuArray::Button::DELETE_BUTTON:
00068                     if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
00069                         break;
00070                     this->array->deleteSquare(
00071                         std::stoi(this->menu->deleteModeValue),
00072                         this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
00073                     );
00074                     std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00075                     this->menu->deleteModeValue = "None";
00076                     this->controlMenu->reset();
00077                     break;
00078                 case constants::MenuArray::Button::UPDATE_BUTTON:
00079                     if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
00080 "None" || this->menu->updateModeValue[0].empty())
00081                         break;
00082
```

```

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 > StaticArrayScene::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](#).

```
00008                                     {
00009     this->window = window;
00010
00011     this->cursor = "|";
00012
00013     this->box.setPosition(position);
00014     this->box.setSize(sf::Vector2f(static_cast<float>((maxLength + 1) * 12), static_cast<float>(size *
00015 1.5)));
00016     this->box.setFillColor(boxColor);
00017     this->box.setOutlineColor(sf::Color::Black);
00018     this->box.setOutlineThickness(1);
00019     this->font.loadFromFile(constants::fontPath);
00020     this->text.setFont(this->font);
00021     this->text.setCharacterSize(size);
00022     this->text.setFillColor(textColor);
00023     this->text.setPosition(position);
00024
00025     this->maxLength = maxLength;
00026     this->textColor = textColor;
00027     this->boxColor = boxColor;
00028
00029     this->cursorVisible = true;
00030     this->flickerClock.restart();
00031 }
```



## 7.41.3 Member Function Documentation

### 7.41.3.1 getBox()

```
sf::RectangleShape TextBox::getBox ( ) const
```

Definition at line 88 of file [Textbox.cpp](#).

```
00088 {
00089     return this->box;
00090 }
```

### 7.41.3.2 getTextString()

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

Definition at line 84 of file [Textbox.cpp](#).

```
00084 {
00085     return this->inputString;
00086 }
```

### 7.41.3.3 pollEvent()

```
void TextBox::pollEvent (
    sf::Event event )
```

Definition at line 33 of file [Textbox.cpp](#).

```
00033 {
00034     if (event.type == sf::Event::TextEntered)
00035     {
00036         if (event.text.unicode == '\b')
00037         {
00038             if (!this->inputString.empty())
00039             {
00040                 this->inputString.pop_back();
00041             }
00042         }
00043         else if (((48 <= event.text.unicode && event.text.unicode <= 57) || event.text.unicode ==
static_cast<int>(' ', ' ')) && this->inputString.size() < this->maxLength)
00044         {
00045             this->inputString += static_cast<char>(event.text.unicode);
00046         }
00047         this->text.setString(this->inputString);
00048     }
00049 }
00050
00051 // if (event.type == sf::Event::Resized)
00052 // {
00053 //     box.setPosition(
00054 //         static_cast<float>(this->window->getSize().x) / 2 - box.getSize().x / 2,
00055 //         static_cast<float>(this->window->getSize().y) / 2 - box.getSize().y / 2
00056 //     );
00057 //     text.setPosition(box.getPosition().x + 10, box.getPosition().y);
00058 //     cursor.setPosition(text.getGlobalBounds().width + text.getPosition().x,
cursor.getPosition().y);
00059 // }
00060 }
```

#### 7.41.3.4 render()

```
void TextBox::render ( )
```

Definition at line 79 of file [Textbox.cpp](#).

```
00079         {
00080             this->window->draw(this->box);
00081             this->window->draw(this->text);
00082         }
```

#### 7.41.3.5 resetInput()

```
void TextBox::resetInput ( )
```

Definition at line 92 of file [Textbox.cpp](#).

```
00092         {
00093             this->inputString = "";
00094             this->text.setString(this->inputString);
00095         }
```

#### 7.41.3.6 update()

```
void TextBox::update ( )
```

Definition at line 62 of file [Textbox.cpp](#).

```
00062         {
00063             if (this->flickerClock.getElapsedTime().asSeconds() >= 0.5)
00064             {
00065                 this->cursorVisible = !this->cursorVisible;
00066                 this->flickerClock.restart();
00067             }
00068
00069             if (this->cursorVisible)
00070             {
00071                 this->text.setString(this->inputString + this->cursor);
00072             }
00073             else
00074             {
00075                 this->text.setString(this->inputString);
00076             }
00077         }
```

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](#).

```
00216 {
00217     this->capacity = capacity;
00218     this->_size = capacity;
00219     this->arr = new T[this->capacity];
00220 }
```

**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](#).

```
00201 {
00202     this->capacity = other.capacity;
00203     this->_size = other._size;
00204     this->arr = new T[this->capacity];
00205     for (int i = 0; i < this->_size; i++) {
00206         this->arr[i] = other.arr[i];
00207     }
00208 }
```

**7.42.2.4 ~Vector()**

```
template<class T >
Vector< T >::~~Vector
```

Definition at line 211 of file [Vector.h](#).

```
00211 {
00212     delete[] this->arr;
00213 }
```

**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](#).

```
00046 {
00047     this->clear();
00048     this->resize(_capacity);
00049     for (int i = 0; i < capacity; ++i) {
00050         this->arr[i] = data;
00051     }
00052
00053 }
```

### 7.42.3.2 at()

```
template<class T >
T & Vector< T >::at (
    int index ) const
```

Definition at line 91 of file [Vector.h](#).

```
00091     {
00092         return this->arr[index];
00093     }
```

### 7.42.3.3 back()

```
template<class T >
T & Vector< T >::back
```

Definition at line 81 of file [Vector.h](#).

```
00081     {
00082         return this->arr[this->_size - 1];
00083     }
```

### 7.42.3.4 begin()

```
template<class T >
T * Vector< T >::begin
```

Definition at line 71 of file [Vector.h](#).

```
00071     {
00072         return this->arr;
00073     }
```

### 7.42.3.5 clear()

```
template<class T >
void Vector< T >::clear
```

Definition at line 144 of file [Vector.h](#).

```
00144     {
00145         this->_size = 0;
00146     }
```

### 7.42.3.6 data()

```
template<class T >
T * Vector< T >::data
```

Definition at line 76 of file [Vector.h](#).

```
00076     {
00077         return this->arr;
00078     }
```

### 7.42.3.7 empty()

```
template<class T >
bool Vector< T >::empty
```

Definition at line 96 of file [Vector.h](#).

```
00096 {
00097     return this->_size == 0;
00098 }
```

### 7.42.3.8 end()

```
template<class T >
T * Vector< T >::end
```

Definition at line 66 of file [Vector.h](#).

```
00066 {
00067     return this->arr + this->_size;
00068 }
```

### 7.42.3.9 erase() [1/2]

```
template<class T >
void Vector< T >::erase (
    int index )
```

Definition at line 149 of file [Vector.h](#).

```
00149 {
00150     if (index >= 0 && index < this->_size) {
00151         for (int i = index; i < this->_size - 1; i++) {
00152             this->arr[i] = this->arr[i + 1];
00153         }
00154         this->_size--;
00155     }
00156 }
```

### 7.42.3.10 erase() [2/2]

```
template<class T >
void Vector< T >::erase (
    T * position )
```

Definition at line 56 of file [Vector.h](#).

```
00056 {
00057     for (int i = 0; i < this->_size; ++i) {
00058         if (this->arr + i == position) {
00059             this->erase(i);
00060             break;
00061         }
00062     }
00063 }
```

**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](#).

```
00125     {
00126     return this->arr[index];
00127 }
```

### 7.42.3.16 pop\_back()

```
template<class T >
void Vector< T >::pop_back
```

Definition at line 179 of file [Vector.h](#).

```
00179     {
00180     if (this->_size > 0) {
00181         this->_size--;
00182     }
00183 }
```

### 7.42.3.17 push\_back()

```
template<class T >
void Vector< T >::push_back (
    T data )
```

Definition at line 186 of file [Vector.h](#).

```
00186     {
00187     if (this->_size >= this->capacity) {
00188         this->capacity *= 2;
00189         T* temp = new T[this->capacity];
00190         for (int i = 0; i < this->_size; i++) {
00191             temp[i] = this->arr[i];
00192         }
00193         delete[] this->arr;
00194         this->arr = temp;
00195     }
00196     this->arr[this->_size] = data;
00197     this->_size++;
00198 }
```

### 7.42.3.18 resize()

```
template<class T >
void Vector< T >::resize (
    int capacity )
```

Definition at line 130 of file [Vector.h](#).

```
00130     {
00131     this->_size = _capacity;
00132     if (_capacity > 0) {
00133         this->capacity = _capacity;
00134         T* temp = new T[this->capacity];
00135         for (int i = 0; i < this->_size; i++) {
00136             temp[i] = this->arr[i];
00137         }
00138         delete[] this->arr;
00139         this->arr = temp;
00140     }
00141 }
```



## 7.42.3.19 size()

```
template<class T >
int Vector< T >::size
```

Definition at line 101 of file [Vector.h](#).

```
00101 {
00102     return this->_size;
00103 }
```

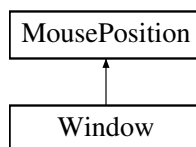
The documentation for this class was generated from the following file:

- [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](#).

```
00050     {
00051         this->initWindow();
00052         this->initScenes();
00053         this->init();
00054     }
```

#### 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](#).

```
00060     {
00061         // event polling
00062         while (this->window->pollEvent(this->event)) {
00063             switch (this->event.type) {
00064                 case sf::Event::Closed:
00065                     this->window->close();
00066                     break;
00067                 case sf::Event::KeyPressed:
00068                     if (this->event.key.code == sf::Keyboard::Q) {
00069                         std::cout << "You have pressed Q!\n";
00070                     }
00071                     if (this->event.key.code == sf::Keyboard::W) {
00072                         std::cout << "You have pressed W!\n";
00073                     }
00074                     break;
00075                 default:
00076                     break;
00077             }
00078
00079             if (this->submenuButton->pollEvent(this->mousePosView)) {
00080                 std::cout << "You have clicked on submenu button!\n";
00081                 this->scenes[this->currentScene]->isMenuOpen = (this->submenuButton->getTextString() ==
00082                     "<");
00083             }
```

```

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\_SCENE  
,  
constants::sceneVariables::STACK\_SCENE , constants::sceneVariables::QUEUE\_SCENE , constants::sceneVariables::STATIC\_SCENE  
, 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\_BUTTON  
, 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::DELETE\_BUTTON  
, 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\_BUTTON  
, 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.](#)

```

00001 //
00002 // Created by dirii on 23/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_CONSTANTS_HPP
00006 #define VISUALGO_CS162_CONSTANTS_HPP
00007
00008 #include <SFML/Graphics.hpp>
00009
00010 namespace constants{
00011     namespace sceneVariables {
00012         constexpr int SCENE_COUNT = 8;
00013         enum Scene {
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         };
00023         constexpr char SCENE_NAMES[SCENE_COUNT][50] = {
00024             "Main Menu",
00025             "Singly Linked List",
00026             "Doubly Linked List",
00027             "Circular Linked List",
00028             "Stack",
00029             "Queue",
00030             "Static Array",
00031             "Dynamic Array",
00032         };
00033         constexpr char NAME_MODE_BUTTON[SCENE_COUNT][50] = {
00034             "Main Menu",

```



```

00035         "SLL",
00036         "DLL",
00037         "CLL",
00038         "Stack",
00039         "Queue",
00040         "Static Array",
00041         "Dynamic Array"
00042     };
00043 }
00044
00045 namespace MenuArray{
00046     enum class Type{
00047         DYNAMIC,
00048         STATIC
00049     };
00050
00051     constexpr int BUTTON_COUNT = 6;
00052     enum Button{
00053         CREATE_BUTTON,
00054         ADD_BUTTON,
00055         DELETE_BUTTON,
00056         UPDATE_BUTTON,
00057         SEARCH_BUTTON,
00058         ALLOCATE_BUTTON,
00059         NONE
00060     };
00061     constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
00062         "Create",
00063         "Add",
00064         "Delete",
00065         "Update",
00066         "Search",
00067         "Allocate"
00068     };
00069     constexpr int BUTTON_NAME_SIZE = 15;
00070
00071     namespace CreateMode {
00072         constexpr int BUTTON_COUNT = 3;
00073         enum Button {
00074             RANDOM_BUTTON,
00075             DEFINED_LIST_BUTTON,
00076             FILE_BUTTON,
00077             NONE
00078         };
00079         constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
00080             "Random",
00081             "Defined List",
00082             "File"
00083         };
00084         constexpr int NAME_SIZE = 15;
00085
00086         constexpr int TEXTBOX_COUNT = 2;
00087         constexpr char TEXTBOX_NAMES[2][50] = {
00088             "Amount = ",
00089             "List = "
00090         };
00091
00092         constexpr int TEXTBOX_LENGTH[2] = {
00093             2,
00094             30 // for input a defined list
00095         };
00096     }
00097     namespace AddMode {
00098         constexpr int TEXTBOX_COUNT = 2;
00099         constexpr char TEXTBOX_NAMES[2][50] = {
00100             "Position = ",
00101             "Value = "
00102         };
00103         constexpr int TEXTBOX_LENGTH[2] = {
00104             2,
00105             2
00106         };
00107         enum Textbox{
00108             POSITION_TEXTBOX,
00109             VALUE_TEXTBOX,
00110             NONE
00111         };
00112     };
00113     namespace DeleteMode {
00114         constexpr int TEXTBOX_COUNT = 1;
00115         constexpr char TEXTBOX_NAME[50] = "Position = ";
00116         constexpr int TEXTBOX_LENGTH = 2;
00117         enum Textbox{
00118             POSITION_TEXTBOX,
00119             NONE
00120         };
00121     }

```

```

00122     namespace UpdateMode {
00123         constexpr int TEXTBOX_COUNT = 2;
00124         constexpr char TEXTBOX_NAMES[2][50] = {
00125             "Position = ",
00126             "Value = "
00127         };
00128         constexpr int TEXTBOX_LENGTH[2] = {
00129             2,
00130             2
00131         };
00132         enum Textbox{
00133             POSITION_TEXTBOX,
00134             VALUE_TEXTBOX,
00135             NONE
00136         };
00137     }
00138     namespace SearchMode {
00139         constexpr int TEXTBOX_COUNT = 1;
00140         constexpr char TEXTBOX_NAME[50] = "Value = ";
00141         constexpr int TEXTBOX_LENGTH = 2;
00142         enum Textbox{
00143             VALUE_TEXTBOX,
00144             NONE
00145         };
00146     }
00147     namespace AllocateMode {
00148         constexpr int TEXTBOX_COUNT = 1;
00149         constexpr char TEXTBOX_NAME[50] = "Size = ";
00150         constexpr int TEXTBOX_LENGTH = 2;
00151         enum Textbox{
00152             VALUE_TEXTBOX,
00153             NONE
00154         };
00155     }
00156 };
00157
00158 namespace MenuDataStructure{
00159     constexpr int BUTTON_COUNT = 4;
00160     enum Button{
00161         CREATE_BUTTON,
00162         PUSH_BUTTON,
00163         POP_BUTTON,
00164         CLEAR_BUTTON,
00165         NONE
00166     };
00167     constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
00168         "Create",
00169         "Push",
00170         "Pop",
00171         "Clear"
00172     };
00173     constexpr int BUTTON_NAME_SIZE = 15;
00174
00175     namespace CreateMode {
00176         constexpr int BUTTON_COUNT = 3;
00177         enum Button {
00178             RANDOM_BUTTON,
00179             DEFINED_LIST_BUTTON,
00180             FILE_BUTTON,
00181             NONE
00182         };
00183         constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
00184             "Random",
00185             "Defined List",
00186             "File"
00187         };
00188         constexpr int NAME_SIZE = 15;
00189
00190         constexpr int TEXTBOX_COUNT = 2;
00191         constexpr char TEXTBOX_NAMES[2][50] = {
00192             "Amount = ",
00193             "List = "
00194         };
00195
00196         constexpr int TEXTBOX_LENGTH[2] = {
00197             2,
00198             30 // for input a defined list
00199         };
00200     }
00201     namespace PushMode{
00202         constexpr int TEXTBOX_COUNT = 1;
00203         constexpr char TEXTBOX_NAME[50] = "Value = ";
00204         constexpr int TEXTBOX_LENGTH = 2;
00205         enum Textbox{
00206             VALUE_TEXTBOX,
00207             NONE
00208         };

```

```

00209     }
00210 }
00211
00212 namespace MenuLinkedList {
00213     constexpr int BUTTON_COUNT = 5;
00214     enum Button {
00215         CREATE_BUTTON,
00216         ADD_BUTTON,
00217         DELETE_BUTTON,
00218         UPDATE_BUTTON,
00219         SEARCH_BUTTON,
00220         NONE
00221     };
00222     constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
00223         "Create",
00224         "Add",
00225         "Delete",
00226         "Update",
00227         "Search"
00228     };
00229     constexpr int BUTTON_NAME_SIZE = 15;
00230
00231     namespace CreateMode {
00232         constexpr int BUTON_COUNT = 3;
00233         enum Button {
00234             RANDOM_BUTTON,
00235             DEFINED_LIST_BUTTON,
00236             FILE_BUTTON,
00237             NONE
00238         };
00239         constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
00240             "Random",
00241             "Defined List",
00242             "File"
00243         };
00244         constexpr int NAME_SIZE = 15;
00245
00246         constexpr int TEXTBOX_COUNT = 2;
00247         constexpr char TEXTBOX_NAMES[2][50] = {
00248             "Amount = ",
00249             "List = "
00250         };
00251
00252         constexpr int TEXTBOX_LENGTH[2] = {
00253             2,
00254             30 // for input a defined list
00255         };
00256     }
00257     namespace AddMode{
00258         constexpr int TEXTBOX_COUNT = 2;
00259         constexpr char TEXTBOX_NAMES[2][50] = {
00260             "Position = ",
00261             "Value = "
00262         };
00263         constexpr int TEXTBOX_LENGTH[2] = {
00264             2,
00265             2
00266         };
00267         enum Textbox{
00268             POSITION_TEXTBOX,
00269             VALUE_TEXTBOX,
00270             NONE
00271         };
00272     }
00273     namespace DeleteMode{
00274         constexpr int TEXTBOX_COUNT = 1;
00275         constexpr char TEXTBOX_NAME[50] = "Position = ";
00276         constexpr int TEXTBOX_LENGTH = 2;
00277         enum Textbox{
00278             POSITION_TEXTBOX,
00279             NONE
00280         };
00281     }
00282     namespace UpdateMode{
00283         constexpr int TEXTBOX_COUNT = 2;
00284         constexpr char TEXTBOX_NAMES[2][50] = {
00285             "Position = ",
00286             "Value = "
00287         };
00288         constexpr int TEXTBOX_LENGTH[2] = {
00289             2,
00290             2
00291         };
00292         enum Textbox{
00293             POSITION_TEXTBOX,
00294             VALUE_TEXTBOX,
00295             NONE

```

```

00296         };
00297     }
00298     namespace SearchMode{
00299         constexpr int TEXTBOX_COUNT = 1;
00300         constexpr char TEXTBOX_NAME[50] = "Value = ";
00301         constexpr int TEXTBOX_LENGTH = 2;
00302         enum Textbox{
00303             VALUE_TEXTBOX,
00304             NONE
00305         };
00306     }
00307 }
00308
00309 namespace NodeInfo{
00310     static float radius = 30,
00311                 outlineThickness = 2;
00312     static int pointCount = 200,
00313               fontSize = 20;
00314     static sf::Vector2f originNode(100, 300);
00315     static float offsetX = 170,
00316                 offsetY = 150;
00317 }
00318
00319 namespace Square{
00320     static float length = 60,
00321                 outlineThickness = 6;
00322     static int fontSize = 20;
00323     static sf::Vector2f originNode(100, 300);
00324     static float offsetX = 72,
00325                 offsetY = 150;
00326 }
00327
00328 namespace Arrow{
00329     static sf::Vector2i sizeArrow(752, 214),
00330                           sizeRectangle(192, 37);
00331     static sf::Vector2f defaultScaleArrow(0.2f, 0.15f),
00332                           defaultScaleRectangle(0.6f, 0.16f);
00333 }
00334
00335 namespace ControlMenu{
00336     enum class Button{
00337         PREVIOUS,
00338         PLAY,
00339         NEXT,
00340         SPEED_DOWN,
00341         SPEED_UP,
00342         None
00343     };
00344
00345     constexpr int BUTTON_COUNT = 5,
00346                 BUTTON_NAME_SIZE = 15,
00347                 TEXT_SIZE = 15;
00348     constexpr char BUTTON_NAMES[BUTTON_COUNT][50] = {
00349         "<",
00350         "[=]",
00351         ">",
00352         "<<",
00353         ">>"
00354     };
00355
00356     static sf::Vector2f buttonSize(50, 50);
00357     static float coordinateY = 930,
00358                 middleX = 1760 / 2.0f - buttonSize.x / 2.0f,
00359                 leftX = 1760 / 7.0f;
00360     static sf::Vector2f buttonPos[5] = {
00361         sf::Vector2f(middleX - 2 * buttonSize.x, coordinateY),
00362         sf::Vector2f(middleX, coordinateY),
00363         sf::Vector2f(middleX + 2 * buttonSize.x, coordinateY),
00364         sf::Vector2f(leftX, coordinateY),
00365         sf::Vector2f(leftX + 3 * buttonSize.x, coordinateY)
00366     };
00367 }
00368
00369 namespace Highlighter{
00370     static sf::Vector2f codePos(1726, 930),
00371                       codeScale(0.6f, 0.6f);
00372
00373     namespace SLL{
00374         const std::pair<const char*, const int> CODES_PATH[4] = {
00375             std::make_pair("../assets/code/SLL/add.png", 10),
00376             std::make_pair("../assets/code/SLL/delete.png", 11),
00377             std::make_pair("../assets/code/SLL/update.png", 4),
00378             std::make_pair("../assets/code/SLL/search.png", 6)
00379         };
00380     }
00381
00382     namespace DLL{

```

```

00383         const std::pair<const char*, const int> CODES_PATH[] = {
00384             std::make_pair("../assets/code/DLL/add_beginning.png", 8),
00385             std::make_pair("../assets/code/DLL/add_ending.png", 5),
00386             std::make_pair("../assets/code/DLL/add_middle.png", 9),
00387             std::make_pair("../assets/code/DLL/delete_beginning.png", 8),
00388             std::make_pair("../assets/code/DLL/delete_ending.png", 5),
00389             std::make_pair("../assets/code/DLL/delete_middle.png", 7),
00390             std::make_pair("../assets/code/DLL/update.png", 4),
00391             std::make_pair("../assets/code/DLL/search.png", 6)
00392         };
00393     }
00394 }
00395
00396 namespace LinkedList{
00397     static float DELAY_TIME = 1.0f;
00398 }
00399
00400 namespace TitleNode{
00401     static int fontSize = 20;
00402     static float offsetY = 50;
00403 }
00404
00405 // information of window
00406 static int Width = 1760,
00407         Height = 992;
00408 constexpr char titleWindow[] = "Visualgo CS162 - Phan Minh Quang";
00409 static int fps = 144;
00410
00411 constexpr char fontPath[] = "../assets/fonts/Hack_reg.ttf";
00412
00413 // colors
00414 static sf::Color normalGreen(189, 210, 182),
00415                 hoverGreen(162, 178, 159),
00416                 clickGreen(121, 135, 119),
00417                 transparentGreen(189, 210, 182, 150);
00418
00419 static sf::Color normalGray(200, 200, 200),
00420                 hoverGray(150, 150, 150),
00421                 clickGray(100, 100, 100);
00422
00423 static sf::Color normalOrange(255, 145, 77);
00424
00425 static sf::Color titleGreen(64, 81, 59);
00426
00427 // positions of buttons
00428 static sf::Vector2f submenuButtonPos = sf::Vector2f(0, 690),
00429                 demoCodeButtonPos = sf::Vector2f(1736, 690),
00430                 modeButtonPos = sf::Vector2f(10, 10);
00431
00432 // size of buttons
00433 static sf::Vector2f sideButtonSize = sf::Vector2f(24, 200),
00434                 modeButtonSize = sf::Vector2f(150, 40),
00435                 optionButtonSize = sf::Vector2f(130, sideButtonSize.y / static_cast<float>(5)
- 1),
00436                 goButtonSize = sf::Vector2f(50, 30);
00437 static float distance2ModeButtons = 10;
00438
00439 // size text of buttons
00440 static int sizeTextModeButton = 15;
00441
00442 // rounding button
00443 static int CORNER_POINT_COUNT_BUTTON = 15;
00444 static float CORNER_RADIUS_BUTTON = 5;
00445 }
00446
00447 #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.](#)

```

00001 //
00002 // Created by dirii on 28/04/2023.
00003 //
00004
00005 #include "Array.hpp"
00006
00007 #include <utility>
00008
00009 Array::Array(sf::RenderWindow *window, TypeArray typeArray) : BaseDraw(window) {
00010     this->init(typeArray);
00011     this->createArray(0);
00012 }
00013
00014 Array::Array(sf::RenderWindow *window, Array::TypeArray typeArray, int size) : BaseDraw(window) {
00015     this->init(typeArray);
00016     this->createArray(size);
00017 }
00018
00019 Array::Array(sf::RenderWindow *window, Array::TypeArray typeArray, std::vector<std::string> values) :
    BaseDraw(window) {
00020     this->init(typeArray);
00021     this->createArray(std::move(values));
00022 }
00023
00024 void Array::init(Array::TypeArray typeArray) {
00025     this->typeArray = typeArray;
00026     this->highlighter = nullptr;
00027     this->delayTime = constants::LinkedList::DELAY_TIME;
00028     this->size = 0;
00029 }
00030
00031 void Array::render() {
00032     for (auto &square : this->squares) {
00033         square->render();
00034     }
00035     for (auto &square : this->squaresTemp) {
00036         square->render();
00037     }
00038 }
00039
00040 void Array::renderHighlighter() {
00041     if (this->highlighter)
00042         this->highlighter->render();
00043 }
00044
00045 void Array::update() {
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 }
00050
00051 void Array::setSpeed(float _speed) {
00052     this->speed = _speed;
00053 }
00054
00055 int Array::findValue(const std::string &value) {
00056     for (int i = 0; i < this->size; i++) {
00057         if (this->squares[i]->getValue() == value)
00058             return i;
00059     }
00060     return this->size;
00061 }
00062
00063 int Array::getSize() const {
00064     return this->size;
00065 }
00066
00067 void Array::processControlMenu(ControlMenu::StatusCode status) {
00068     if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)
00069         return;
00070     switch (status){
00071         case ControlMenu::StatusCode::PREVIOUS:
00072             if (this->currentEvent > 0)
00073                 --this->currentEvent;
00074             break;
00075         case ControlMenu::StatusCode::PAUSE:
00076             // std::cout << "PAUSE" << std::endl;
00077             break;
00078         case ControlMenu::StatusCode::PLAY:
00079             if (this->currentEvent + 1 < this->events.size()) {
00080                 this->isDelay = true;
00081                 this->clock.restart();
00082             }
00083         case ControlMenu::StatusCode::NEXT:
00084             if (this->currentEvent + 1 < this->events.size())
00085                 ++this->currentEvent;

```

```

00086         break;
00087     default:
00088         break;
00089     }
00090 }
00091
00092 void Array::initHighlighter(int linesCount, const char *codePath) {
00093     delete this->highlighter;
00094     this->highlighter = new Highlighter(
00095         this->window,
00096         linesCount,
00097         codePath
00098     );
00099 }
00100
00101 void Array::toggleLines(std::vector<int> lines) {
00102     this->highlighter->toggle(std::move(lines));
00103 }
00104
00105 void Array::updateAnimation() {
00106     if (this->squares.empty())
00107         return;
00108     for (auto &square : this->squares) {
00109         square->reset();
00110     }
00111     for (auto &square : this->squaresTemp) {
00112         square->reset();
00113     }
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 }
00138
00139 void Array::resetEvents() {
00140     delete this->highlighter;
00141     this->highlighter = nullptr;
00142     this->currentEvent = 0;
00143     this->events.clear();
00144     this->squaresTemp.clear();
00145
00146     while (!this->squares.empty() && this->squares.back()->getStatus() == Square::Status::hidden)
00147         this->squares.pop_back();
00148
00149     for (int i = 0; i < this->size; ++i)
00150         this->squares[i]->setStatus(Square::Status::active);
00151     for (int i = this->size; i < this->squares.size(); ++i)
00152         this->squares[i]->setStatus(Square::Status::inactive);
00153     if (this->size)
00154         this->squares[this->size - 1]->setTitle("n");
00155 }
00156
00157 void Array::createArray(int _size) {
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) *
00169                 constants::Square::offsetX,
00170                 constants::Square::originNode.y
00171             )
00172         );

```

```

00172         this->squares[i]->setStatus(Square::Status::active);
00173     }
00174     if (this->size)
00175         this->squares[this->size - 1]->setTitle("n");
00176 }
00177
00178 void Array::createArray(const std::vector<std::string>& values) {
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) *
00190                 constants::Square::offsetX,
00191                 constants::Square::originNode.y
00192             ));
00193         this->squares[i]->setStatus(Square::Status::active);
00194     }
00195     if (this->size)
00196         this->squares[this->size - 1]->setTitle("n");
00197 }
00198
00199 int Array::getSquaresSize() const {
00200     return (int)this->squares.size();
00201 }
00202
00203 void Array::allocateSquare(int _size, const std::vector<EventAnimation> &listEvents) {
00204     this->squaresTemp.resize(_size);
00205     // this->squares.resize();
00206
00207     while (this->squares.size() < _size)
00208         this->squares.push_back(new SquareInfo(
00209             this->window,
00210             "",
00211             sf::Vector2f(
00212                 constants::Square::originNode.x + static_cast<float>(this->squares.size()) *
00213                 constants::Square::offsetX,
00214                 constants::Square::originNode.y
00215             ));
00216
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) *
00223                 constants::Square::offsetX,
00224                 constants::Square::originNode.y + constants::Square::offsetY
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 }
00233
00234 void Array::addSquare(int position, std::string value, const std::vector<EventAnimation> &listEvents)
00235 {
00236     if (position < 0 || position > this->size)
00237         return;
00238     ++this->size;
00239     if (this->typeArray == TypeArray::DYNAMIC && this->size > this->getSquaresSize()) {
00240         this->squares.push_back(new SquareInfo(
00241             this->window,
00242             "",
00243             sf::Vector2f(
00244                 constants::Square::originNode.x + static_cast<float>(this->getSquaresSize()) *
00245                 constants::Square::offsetX,
00246                 constants::Square::originNode.y
00247             ));
00248         this->squaresTemp.resize(this->size);
00249         for (int i = 0; i < this->size; ++i) {
00250             this->squaresTemp[i] = new SquareInfo(
00251                 this->window,
00252                 "",
00253                 sf::Vector2f(

```



```

00254         constants::Square::originNode.x + static_cast<float>(i) *
constants::Square::offsetX,
00255         constants::Square::originNode.y + constants::Square::offsetY
00256     )
00257 );
00258     this->squaresTemp[i]->setValue(this->squares[i]->getValue());
00259 }
00260 }
00261
00262     if (size > this->getSquaresSize())
00263         --this->size;
00264
00265     for (int i = this->size - 1; i > position; --i)
00266         this->squares[i]->setValue(this->squares[i - 1]->getValue());
00267     this->squares[position]->setValue(std::move(value));
00268     for (int i = 0; i < position; ++i)
00269         this->squares[i]->setValue(this->squares[i]->getValue());
00270
00271     this->currentEvent = 0;
00272     this->events = listEvents;
00273 }
00274
00275 void Array::deleteSquare(int position, const std::vector<EventAnimation> &listEvents) {
00276     if (position < 0 || position >= this->size)
00277         return;
00278     --this->size;
00279
00280     for (int i = position; i < this->size; ++i)
00281         this->squares[i]->setValue(this->squares[i + 1]->getValue());
00282     for (int i = 0; i < position; ++i)
00283         this->squares[i]->setValue(this->squares[i]->getValue());
00284     this->squares[this->size]->setValue(this->squares[this->size]->getValue());
00285
00286     this->currentEvent = 0;
00287     this->events = listEvents;
00288 }
00289
00290 void Array::updateSquare(int position, std::string value, const std::vector<EventAnimation>
&listEvents) {
00291     if (position < 0 || position >= this->size)
00292         return;
00293     this->squares[position]->setValue(std::move(value));
00294
00295     this->currentEvent = 0;
00296     this->events = listEvents;
00297 }
00298
00299 void Array::searchSquare(const std::vector<EventAnimation> &listEvents) {
00300     this->currentEvent = 0;
00301     this->events = listEvents;
00302 }
00303
00304 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 28/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_ARRAY_HPP
00006 #define VISUALGO_CS162_ARRAY_HPP
00007
00008 #include "Random.h"
00009 #include "core/Vector.h"
00010 #include "draw/SquareInfo.hpp"
00011 #include "libScene/Highlighter.hpp"
00012 #include "libScene/ControlMenu.hpp"
00013 #include "EventAnimation.hpp"
00014
00015 class Array : public BaseDraw{
00016 public:
00017     enum class TypeArray{
00018         DYNAMIC,
00019         STATIC
00020     };
00021
00022     Array(sf::RenderWindow *window, TypeArray typeArray);
00023     Array(sf::RenderWindow *window, TypeArray typeArray, int size);
00024     Array(sf::RenderWindow *window, TypeArray typeArray, std::vector<std::string> values);
00025     void init(TypeArray typeArray);
00026     ~Array() = default;
00027     void render() override;
00028     void renderHighlighter();
00029     void update();
00030
00031     void setSpeed(float speed);
00032     int findValue(const std::string& value);
00033
00034     void updateAnimation();
00035     void resetEvents();
00036
00037     [[nodiscard]] int getSize() const;
00038     [[nodiscard]] int getSquaresSize() const;
00039
00040     void processControlMenu(ControlMenu::StatusCode status);
00041
00042     // operations of highlighter
00043     void initHighlighter(int linesCount, const char *codePath);
00044     void toggleLines(std::vector<int> lines);
00045
00046     // operations of array
00047     void createArray(int size);
00048     void createArray(const std::vector<std::string>& values);
00049     void allocateSquare(int size, const std::vector<EventAnimation>& listEvents);
00050     void addSquare(int position, std::string value, const std::vector<EventAnimation>& listEvents);
00051     void deleteSquare(int position, const std::vector<EventAnimation>& listEvents);
00052     void updateSquare(int position, std::string value, const std::vector<EventAnimation>& listEvents);
00053     void searchSquare(const std::vector<EventAnimation>& listEvents);
00054
00055 private:
00056     sf::Clock clock;
00057     int chosenNode = 0, deletedNode = -1;
00058     TypeArray typeArray;
00059
00060     Vector<SquareInfo*> squares, squaresTemp;
00061     int size;
00062
00063     Highlighter* highlighter;
00064
00065     std::vector<EventAnimation> events;
00066     int currentEvent = 0;
00067
00068     float speed, delayTime;
00069     bool isDelay = false;
00070 };
00071
00072 #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.](#)

```

00001 //
00002 // Created by dirii on 16/04/2023.
00003 //
00004
00005 #include "EventAnimation.hpp"
00006
00007 EventAnimation::EventAnimation() {
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 }
00021
00022 void EventAnimation::reset() {
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 }
00036
00037 EventAnimation::~EventAnimation() {
00038     this->titleNodes = {};
00039     this->colorArrows = {};
00040     this->hiddenArrows = {};
00041     this->colorNodes = {};
00042     this->lines = {};
00043
00044     this->eventSquares = {};
00045     this->eventSquaresTemp = {};
00046 }
00047
00048 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.](#)

```

00001 //

```

```

00002 // Created by dirii on 16/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_EVENTANIMATION_HPP
00006 #define VISUALGO_CS162_EVENTANIMATION_HPP
00007
00008 #include "draw/NodeInfo.hpp"
00009 #include "draw/SquareInfo.hpp"
00010
00011 struct EventSquare{
00012     Square::Status status = Square::Status::inactive;
00013     bool isPrintPreVal = false;
00014     std::string title{};
00015
00016     EventSquare() = default;
00017     ~EventSquare() = default;
00018 };
00019
00020 class EventAnimation{
00021 public:
00022     // for linked list
00023     std::vector<std::pair<int, std::string>> titleNodes;
00024     std::vector<std::pair<int, NodeInfo::ArrowType>> colorArrows;
00025     std::vector<std::pair<int, NodeInfo::ArrowType>> hiddenArrows;
00026     std::vector<int> colorNodes;
00027     NodeInfo::StatusNode statusChosenNode;
00028     bool isPrintPreVal, isPrintNormal, isShowBackArrow;
00029     std::pair<int, int> indexBackArrow;
00030
00031     // for array
00032     std::vector<EventSquare> eventSquares{}, eventSquaresTemp{};
00033
00034     std::vector<int> lines;
00035
00036     EventAnimation();
00037     ~EventAnimation();
00038
00039     EventAnimation& operator=(const EventAnimation& other);
00040
00041     void reset();
00042 };
00043
00044 #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.](#)

```
00001 //  
00002 // Portable File Dialogs  
00003 //  
00004 // Copyright 1' 20182022 Sam Hocevar <sam@hocevar.net>  
00005 //  
00006 // This library is free software. It comes without any warranty, to  
00007 // the extent permitted by applicable law. You can redistribute it  
00008 // and/or modify it under the terms of the Do What the Fuck You Want  
00009 // to Public License, Version 2, as published by the WTFPL Task Force.  
00010 // See http://www.wtfpl.net/ for more details.  
00011 //  
00012 //  
00013 #pragma once  
00014 //  
00015 #if _WIN32  
00016 #ifndef WIN32_LEAN_AND_MEAN  
00017 # define WIN32_LEAN_AND_MEAN 1  
00018 #endif  
00019 #include <windows.h>  
00020 #include <commdlg.h>  
00021 #include <shlobj.h>  
00022 #include <shobjidl.h> // IFileDialog  
00023 #include <shellapi.h>  
00024 #include <strsafe.h>  
00025 #include <future>      // std::async  
00026 #include <userenv.h>   // GetUserProfileDirectory()  
00027 //  
00028 #elif __EMSCRIPTEN__  
00029 #include <emscripten.h>  
00030 //  
00031 #else  
00032 #ifndef _POSIX_C_SOURCE  
00033 # define _POSIX_C_SOURCE 2 // for popen()  
00034 #endif  
00035 #ifdef __APPLE__  
00036 # ifnndef _DARWIN_C_SOURCE  
00037 # define _DARWIN_C_SOURCE  
00038 # endif  
00039 #endif  
00040 #include <stdio.h>      // popen()  
00041 #include <stdlib.h>     // std::getenv()  
00042 #include <fcntl.h>      // fcntl()  
00043 #include <unistd.h>     // read(), pipe(), dup2(), getuid()  
00044 #include <signal.h>     // ::kill, std::signal  
00045 #include <sys/stat.h>   // stat()  
00046 #include <sys/wait.h>   // waitpid()  
00047 #include <pwd.h>        // getpwnam()  
00048 #endif  
00049 //  
00050 #include <string>        // std::string  
00051 #include <memory>       // std::shared_ptr
```

```

00052 #include <iostream> // std::ostream
00053 #include <map>       // std::map
00054 #include <set>       // std::set
00055 #include <regex>     // std::regex
00056 #include <thread>    // std::mutex, std::this_thread
00057 #include <chrono>    // std::chrono
00058
00059 // Versions of mingw64 g++ up to 9.3.0 do not have a complete IFFileDialog
00060 #ifndef PFD_HAS_IFILEDIALOG
00061 #   define PFD_HAS_IFILEDIALOG 1
00062 #   if (defined __MINGW64__ || defined __MINGW32__) && defined __GXX_ABI_VERSION
00063 #       if __GXX_ABI_VERSION <= 1013
00064 #           undef PFD_HAS_IFILEDIALOG
00065 #           define PFD_HAS_IFILEDIALOG 0
00066 #       endif
00067 #   endif
00068 #endif
00069
00070 namespace pfd
00071 {
00072
00073     enum class button
00074     {
00075         cancel = -1,
00076         ok,
00077         yes,
00078         no,
00079         abort,
00080         retry,
00081         ignore,
00082     };
00083
00084     enum class choice
00085     {
00086         ok = 0,
00087         ok_cancel,
00088         yes_no,
00089         yes_no_cancel,
00090         retry_cancel,
00091         abort_retry_ignore,
00092     };
00093
00094     enum class icon
00095     {
00096         info = 0,
00097         warning,
00098         error,
00099         question,
00100     };
00101
00102 // Additional option flags for various dialog constructors
00103     enum class opt : uint8_t
00104     {
00105         none = 0,
00106         // For file open, allow multiselect.
00107         multiselect = 0x1,
00108         // For file save, force overwrite and disable the confirmation dialog.
00109         force_overwrite = 0x2,
00110         // For folder select, force path to be the provided argument instead
00111         // of the last opened directory, which is the Microsoft-recommended,
00112         // user-friendly behaviour.
00113         force_path = 0x4,
00114     };
00115
00116     inline opt operator |(opt a, opt b) { return opt(uint8_t(a) | uint8_t(b)); }
00117     inline bool operator &(opt a, opt b) { return bool(uint8_t(a) & uint8_t(b)); }
00118
00119 // The settings class, only exposing to the user a way to set verbose mode
00120 // and to force a rescan of installed desktop helpers (zenity, kdialog).
00121     class settings
00122     {
00123     public:
00124         static bool available();
00125
00126         static void verbose(bool value);
00127         static void rescan();
00128
00129     protected:
00130         explicit settings(bool resync = false);
00131
00132         bool check_program(std::string const &program);
00133
00134         inline bool is_osascript() const;
00135         inline bool is_zenity() const;
00136         inline bool is_kdialog() const;
00137
00138         enum class flag

```

```

00139     {
00140         is_scanned = 0,
00141         is_verbose,
00142
00143         has_zenity,
00144         has_matedialog,
00145         has_qarma,
00146         has_kdialog,
00147         is_vista,
00148
00149         max_flag,
00150     };
00151
00152     // Static array of flags for internal state
00153     bool const &flags(flag in_flag) const;
00154
00155     // Non-const getter for the static array of flags
00156     bool &flags(flag in_flag);
00157 };
00158
00159 // Internal classes, not to be used by client applications
00160 namespace internal
00161 {
00162
00163     // Process wait timeout, in milliseconds
00164     static int const default_wait_timeout = 20;
00165
00166     class executor
00167     {
00168     public:
00169         friend class dialog;
00170
00171         // High level function to get the result of a command
00172         std::string result(int *exit_code = nullptr);
00173
00174         // High level function to abort
00175         bool kill();
00176
00177 #if _WIN32
00178         void start_func(std::function<std::string(int *)> const &fun);
00179         static BOOL CALLBACK enum_windows_callback(HWND hwnd, LPARAM lParam);
00180 #elif __EMSCRIPTEN__
00181         void start(int exit_code);
00182 #else
00183         void start_process(std::vector<std::string> const &command);
00184 #endif
00185
00186         ~executor();
00187
00188     protected:
00189         bool ready(int timeout = default_wait_timeout);
00190         void stop();
00191
00192     private:
00193         bool m_running = false;
00194         std::string m_stdout;
00195         int m_exit_code = -1;
00196 #if _WIN32
00197         std::future<std::string> m_future;
00198         std::set<HWND> m_windows;
00199         std::condition_variable m_cond;
00200         std::mutex m_mutex;
00201         DWORD m_tid;
00202 #elif __EMSCRIPTEN__ || __NX__
00203         // FIXME: do something
00204 #else
00205         pid_t m_pid = 0;
00206         int m_fd = -1;
00207 #endif
00208     };
00209
00210     class platform
00211     {
00212     protected:
00213 #if _WIN32
00214         // Helper class around LoadLibraryA() and GetProcAddress() with some safety
00215         class dll
00216         {
00217         public:
00218             dll(std::string const &name);
00219             ~dll();
00220
00221             template<typename T> class proc
00222             {
00223             public:
00224                 proc(dll const &lib, std::string const &sym)
00225                   : m_proc(reinterpret_cast<T *>((void *)::GetProcAddress(lib.handle, sym.c_str())))
```



```

00226         {}
00227
00228         operator bool() const { return m_proc != nullptr; }
00229         operator T *() const { return m_proc; }
00230
00231     private:
00232         T *m_proc;
00233     };
00234
00235     private:
00236         HMODULE handle;
00237     };
00238
00239     // Helper class around CoInitialize() and CoUnInitialize()
00240     class ole32_dll : public dll
00241     {
00242     public:
00243         ole32_dll();
00244         ~ole32_dll();
00245         bool is_initialized();
00246
00247     private:
00248         HRESULT m_state;
00249     };
00250
00251     // Helper class around CreateActCtx() and ActivateActCtx()
00252     class new_style_context
00253     {
00254     public:
00255         new_style_context();
00256         ~new_style_context();
00257
00258     private:
00259         HANDLE create();
00260         ULONG_PTR m_cookie = 0;
00261     };
00262 #endif
00263
00264
00265     class dialog : protected settings, protected platform
00266     {
00267     public:
00268         bool ready(int timeout = default_wait_timeout) const;
00269         bool kill() const;
00270
00271     protected:
00272         explicit dialog();
00273
00274         std::vector<std::string> desktop_helper() const;
00275         static std::string buttons_to_name(choice _choice);
00276         static std::string get_icon_name(icon _icon);
00277
00278         std::string powershell_quote(std::string const &str) const;
00279         std::string osascript_quote(std::string const &str) const;
00280         std::string shell_quote(std::string const &str) const;
00281
00282         // Keep handle to executing command
00283         std::shared_ptr<executor> m_async;
00284     };
00285
00286     class file_dialog : public dialog
00287     {
00288     protected:
00289         enum type
00290         {
00291             open,
00292             save,
00293             folder,
00294         };
00295
00296         file_dialog(type in_type,
00297                     std::string const &title,
00298                     std::string const &default_path = "",
00299                     std::vector<std::string> const &filters = {},
00300                     opt options = opt::none);
00301
00302     protected:
00303         std::string string_result();
00304         std::vector<std::string> vector_result();
00305
00306 #if _WIN32
00307         static int CALLBACK bffcallback(HWND hwnd, UINT uMsg, LPARAM, LPARAM pData);
00308 #if PFD_HAS_IFILEDIALOG
00309         std::string select_folder_vista(IFileDialog *ifd, bool force_path);
00310 #endif
00311
00312         std::wstring m_wtitle;

```

```

00313     std::wstring m_wdefault_path;
00314
00315     std::vector<std::string> m_vector_result;
00316 #endif
00317     };
00318
00319     } // namespace internal
00320
00321     //
00322     // The path class provides some platform-specific path constants
00323     //
00324
00325     class path : protected internal::platform
00326     {
00327     public:
00328         static std::string home();
00329         static std::string separator();
00330     };
00331
00332     //
00333     // The notify widget
00334     //
00335
00336     class notify : public internal::dialog
00337     {
00338     public:
00339         notify(std::string const &title,
00340              std::string const &message,
00341              icon _icon = icon::info);
00342     };
00343
00344     //
00345     // The message widget
00346     //
00347
00348     class message : public internal::dialog
00349     {
00350     public:
00351         message(std::string const &title,
00352              std::string const &text,
00353              choice _choice = choice::ok_cancel,
00354              icon _icon = icon::info);
00355
00356         button result();
00357
00358     private:
00359         // Some extra logic to map the exit code to button number
00360         std::map<int, button> m_mappings;
00361     };
00362
00363     //
00364     // The open_file, save_file, and open_folder widgets
00365     //
00366
00367     class open_file : public internal::file_dialog
00368     {
00369     public:
00370         open_file(std::string const &title,
00371              std::string const &default_path = "",
00372              std::vector<std::string> const &filters = { "All Files", "*" },
00373              opt options = opt::none);
00374
00375     #if defined(__has_cpp_attribute)
00376     #if __has_cpp_attribute(deprecated)
00377         // Backwards compatibility
00378         [[deprecated("Use pfd::opt::multiselect instead of allow_multiselect")]]
00379     #endif
00380     #endif
00381         open_file(std::string const &title,
00382              std::string const &default_path,
00383              std::vector<std::string> const &filters,
00384              bool allow_multiselect);
00385
00386         std::vector<std::string> result();
00387     };
00388
00389     class save_file : public internal::file_dialog
00390     {
00391     public:
00392         save_file(std::string const &title,
00393              std::string const &default_path = "",
00394              std::vector<std::string> const &filters = { "All Files", "*" },
00395              opt options = opt::none);
00396
00397     #if defined(__has_cpp_attribute)
00398     #if __has_cpp_attribute(deprecated)
00399         // Backwards compatibility

```

```

00400     [[deprecated("Use pfd::opt::force_overwrite instead of confirm_overwrite")]]
00401 #endif
00402 #endif
00403     save_file(std::string const &title,
00404              std::string const &default_path,
00405              std::vector<std::string> const &filters,
00406              bool confirm_overwrite);
00407
00408     std::string result();
00409 };
00410
00411 class select_folder : public internal::file_dialog
00412 {
00413 public:
00414     select_folder(std::string const &title,
00415                  std::string const &default_path = "",
00416                  opt options = opt::none);
00417
00418     std::string result();
00419 };
00420
00421 //
00422 // Below this are all the method implementations. You may choose to define the
00423 // macro PFD_SKIP_IMPLEMENTATION everywhere before including this header except
00424 // in one place. This may reduce compilation times.
00425 //
00426 #if !defined PFD_SKIP_IMPLEMENTATION
00427 // internal free functions implementations
00428
00429 namespace internal
00430 {
00431 #if _WIN32
00432     static inline std::wstring str2wstr(std::string const &str)
00433     {
00434         int len = MultiByteToWideChar(CP_UTF8, 0, str.c_str(), (int)str.size(), nullptr, 0);
00435         std::wstring ret(len, '\\0');
00436         MultiByteToWideChar(CP_UTF8, 0, str.c_str(), (int)str.size(), (LPWSTR)ret.data(),
00437                             (int)ret.size());
00438         return ret;
00439     }
00440 #endif
00441
00442     static inline std::string wstr2str(std::wstring const &str)
00443     {
00444         int len = WideCharToMultiByte(CP_UTF8, 0, str.c_str(), (int)str.size(), nullptr, 0, nullptr,
00445                                         nullptr);
00446         std::string ret(len, '\\0');
00447         WideCharToMultiByte(CP_UTF8, 0, str.c_str(), (int)str.size(), (LPSTR)ret.data(), (int)ret.size(),
00448                             nullptr, nullptr);
00449         return ret;
00450     }
00451
00452     static inline bool is_vista()
00453     {
00454         OSVERSIONINFOEXW osv;
00455         memset(&osv, 0, sizeof(osv));
00456         DWORDLONG const mask = VerSetConditionMask(
00457             VerSetConditionMask(
00458                 0, VER_MAJORVERSION, VER_GREATER_EQUAL),
00459             VER_MINORVERSION, VER_GREATER_EQUAL),
00460             VER_SERVICEPACKMAJOR, VER_GREATER_EQUAL);
00461         osv.dwOSVersionInfoSize = sizeof(osv);
00462         osv.dwMajorVersion = HIBYTE(_WIN32_WINNT_VISTA);
00463         osv.dwMinorVersion = LOBYTE(_WIN32_WINNT_VISTA);
00464         osv.wServicePackMajor = 0;
00465
00466         return VerifyVersionInfoW(&osv, VER_MAJORVERSION | VER_MINORVERSION | VER_SERVICEPACKMAJOR, mask)
00467             != FALSE;
00468     }
00469 #endif
00470 // This is necessary until C++20 which will have std::string::ends_with() etc.
00471
00472     static inline bool ends_with(std::string const &str, std::string const &suffix)
00473     {
00474         return suffix.size() <= str.size() &&
00475             str.compare(str.size() - suffix.size(), suffix.size(), suffix) == 0;
00476     }
00477
00478     static inline bool starts_with(std::string const &str, std::string const &prefix)
00479     {
00480         return prefix.size() <= str.size() &&
00481             str.compare(0, prefix.size(), prefix) == 0;
00482     }

```

```

00483
00484 // This is necessary until C++17 which will have std::filesystem::is_directory
00485
00486     static inline bool is_directory(std::string const &path)
00487     {
00488 #if _WIN32
00489         auto attr = GetFileAttributesA(path.c_str());
00490         return attr != INVALID_FILE_ATTRIBUTES && (attr & FILE_ATTRIBUTE_DIRECTORY);
00491 #elif __EMSCRIPTEN__
00492         // TODO
00493         return false;
00494 #else
00495         struct stat s;
00496         return stat(path.c_str(), &s) == 0 && S_ISDIR(s.st_mode);
00497 #endif
00498     }
00499
00500 // This is necessary because getenv is not thread-safe
00501
00502     static inline std::string getenv(std::string const &str)
00503     {
00504 #if _MSC_VER
00505         char *buf = nullptr;
00506         size_t size = 0;
00507         if (_dupenv_s(&buf, &size, str.c_str()) == 0 && buf)
00508         {
00509             std::string ret(buf);
00510             free(buf);
00511             return ret;
00512         }
00513         return "";
00514 #else
00515         auto buf = std::getenv(str.c_str());
00516         return buf ? buf : "";
00517 #endif
00518     }
00519
00520 } // namespace internal
00521
00522 // settings implementation
00523
00524 inline settings::settings(bool resync)
00525 {
00526     flags(flag::is_scanned) &= !resync;
00527
00528     if (flags(flag::is_scanned))
00529         return;
00530
00531     auto pfd_verbose = internal::getenv("PFD_VERBOSE");
00532     auto match_no = std::regex("(0|no|false)", std::regex_constants::icase);
00533     if (!std::regex_match(pfd_verbose, match_no))
00534         flags(flag::is_verbose) = true;
00535
00536 #if _WIN32
00537     flags(flag::is_vista) = internal::is_vista();
00538 #elif !__APPLE__
00539     flags(flag::has_zenity) = check_program("zenity");
00540     flags(flag::has_matedialog) = check_program("matedialog");
00541     flags(flag::has_qarma) = check_program("qarma");
00542     flags(flag::has_kdialog) = check_program("kdialog");
00543
00544     // If multiple helpers are available, try to default to the best one
00545     if (flags(flag::has_zenity) && flags(flag::has_kdialog))
00546     {
00547         auto desktop_name = internal::getenv("XDG_SESSION_DESKTOP");
00548         if (desktop_name == std::string("gnome"))
00549             flags(flag::has_kdialog) = false;
00550         else if (desktop_name == std::string("KDE"))
00551             flags(flag::has_zenity) = false;
00552     }
00553 #endif
00554
00555     flags(flag::is_scanned) = true;
00556 }
00557
00558 inline bool settings::available()
00559 {
00560 #if _WIN32
00561     return true;
00562 #elif __APPLE__
00563     return true;
00564 #elif __EMSCRIPTEN__
00565     // FIXME: Return true after implementation is complete.
00566     return false;
00567 #else
00568     settings tmp;
00569     return tmp.flags(flag::has_zenity) ||

```

```

00570         tmp.flags(flag::has_matedialog) ||
00571         tmp.flags(flag::has_qarma) ||
00572         tmp.flags(flag::has_kdialog);
00573 #endif
00574     }
00575
00576     inline void settings::verbose(bool value)
00577     {
00578         settings().flags(flag::is_verbose) = value;
00579     }
00580
00581     inline void settings::rescan()
00582     {
00583         settings(/* resync = */ true);
00584     }
00585
00586     // Check whether a program is present using which.
00587     inline bool settings::check_program(std::string const &program)
00588     {
00589         #if _WIN32
00590             (void)program;
00591             return false;
00592         #elif __EMSCRIPTEN__
00593             (void)program;
00594             return false;
00595         #else
00596             int exit_code = -1;
00597             internal::executor async;
00598             async.start_process({"bin/sh", "-c", "which " + program});
00599             async.result(&exit_code);
00600             return exit_code == 0;
00601         #endif
00602     }
00603
00604     inline bool settings::is_osascript() const
00605     {
00606         #if __APPLE__
00607             return true;
00608         #else
00609             return false;
00610         #endif
00611     }
00612
00613     inline bool settings::is_zenity() const
00614     {
00615         return flags(flag::has_zenity) ||
00616            flags(flag::has_matedialog) ||
00617            flags(flag::has_qarma);
00618     }
00619
00620     inline bool settings::is_kdialog() const
00621     {
00622         return flags(flag::has_kdialog);
00623     }
00624
00625     inline bool const &settings::flags(flag in_flag) const
00626     {
00627         static bool flags[size_t(flag::max_flag)];
00628         return flags[size_t(in_flag)];
00629     }
00630
00631     inline bool &settings::flags(flag in_flag)
00632     {
00633         return const_cast<bool &>(static_cast<settings const *>(this)->flags(in_flag));
00634     }
00635
00636     // path implementation
00637     inline std::string path::home()
00638     {
00639         #if _WIN32
00640             // First try the USERPROFILE environment variable
00641             auto user_profile = internal::getenv("USERPROFILE");
00642             if (user_profile.size() > 0)
00643                 return user_profile;
00644             // Otherwise, try GetUserProfileDirectory()
00645             HANDLE token = nullptr;
00646             DWORD len = MAX_PATH;
00647             char buf[MAX_PATH] = { '\0' };
00648             if (OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &token))
00649             {
00650                 dll userenv("userenv.dll");
00651                 dll::proc<BOOL WINAPI (HANDLE, LPSTR, LPDWORD)> get_user_profile_directory(userenv,
00652 "GetUserProfileDirectoryA");
00653                 get_user_profile_directory(token, buf, &len);
00654                 CloseHandle(token);
00655                 if (*buf)
00656                     return buf;

```

```

00656     }
00657 #elif __EMSCRIPTEN__
00658     return "/";
00659 #else
00660     // First try the HOME environment variable
00661     auto home = internal::getenv("HOME");
00662     if (home.size() > 0)
00663         return home;
00664     // Otherwise, try getpwuid_r()
00665     size_t len = 4096;
00666 #if defined(_SC_GETPW_R_SIZE_MAX)
00667     auto size_max = sysconf(_SC_GETPW_R_SIZE_MAX);
00668     if (size_max != -1)
00669         len = size_t(size_max);
00670 #endif
00671     std::vector<char> buf(len);
00672     struct passwd pwd, *result;
00673     if (getpwuid_r(getuid(), &pwd, buf.data(), buf.size(), &result) == 0)
00674         return result->pw_dir;
00675 #endif
00676     return "/";
00677 }
00678
00679 inline std::string path::separator()
00680 {
00681 #if _WIN32
00682     return "\\";
00683 #else
00684     return "/";
00685 #endif
00686 }
00687
00688 // executor implementation
00689
00690 inline std::string internal::executor::result(int *exit_code /* = nullptr */)
00691 {
00692     stop();
00693     if (exit_code)
00694         *exit_code = m_exit_code;
00695     return m_stdout;
00696 }
00697
00698 inline bool internal::executor::kill()
00699 {
00700 #if _WIN32
00701     if (m_future.valid())
00702     {
00703         // Close all windows that werent open when we started the future
00704         auto previous_windows = m_windows;
00705         EnumWindows(&enum_windows_callback, (LPARAM)this);
00706         for (auto hwnd : m_windows)
00707             if (previous_windows.find(hwnd) == previous_windows.end())
00708             {
00709                 SendMessage(hwnd, WM_CLOSE, 0, 0);
00710                 // Also send IDNO in case of a Yes/No or Abort/Retry/Ignore messagebox
00711                 SendMessage(hwnd, WM_COMMAND, IDNO, 0);
00712             }
00713     }
00714 #elif __EMSCRIPTEN__ || __NX__
00715     // FIXME: do something
00716     return false; // cannot kill
00717 #else
00718     ::kill(m_pid, SIGKILL);
00719 #endif
00720     stop();
00721     return true;
00722 }
00723
00724 #if _WIN32
00725 inline BOOL CALLBACK internal::executor::enum_windows_callback(HWND hwnd, LPARAM lParam)
00726 {
00727     auto that = (executor *)lParam;
00728
00729     DWORD pid;
00730     auto tid = GetWindowThreadProcessId(hwnd, &pid);
00731     if (tid == that->m_tid)
00732         that->m_windows.insert(hwnd);
00733     return TRUE;
00734 }
00735 #endif
00736
00737 #if _WIN32
00738 inline void internal::executor::start_func(std::function<std::string(int *)> const &fun)
00739 {
00740     stop();
00741
00742     auto trampoline = [fun, this]()

```

```

00743     {
00744         // Save our thread id so that the caller can cancel us
00745         m_tid = GetCurrentThreadId();
00746         EnumWindows(&enum_windows_callback, (LPARAM)this);
00747         m_cond.notify_all();
00748         return fun(&m_exit_code);
00749     };
00750
00751     std::unique_lock<std::mutex> lock(m_mutex);
00752     m_future = std::async(std::launch::async, trampoline);
00753     m_cond.wait(lock);
00754     m_running = true;
00755 }
00756
00757 #elif __EMSCRIPTEN__
00758     inline void internal::executor::start(int exit_code)
00759     {
00760         m_exit_code = exit_code;
00761     }
00762
00763 #else
00764     inline void internal::executor::start_process(std::vector<std::string> const &command)
00765     {
00766         stop();
00767         m_stdout.clear();
00768         m_exit_code = -1;
00769
00770         int in[2], out[2];
00771         if (pipe(in) != 0 || pipe(out) != 0)
00772             return;
00773
00774         m_pid = fork();
00775         if (m_pid < 0)
00776             return;
00777
00778         close(in[m_pid ? 0 : 1]);
00779         close(out[m_pid ? 1 : 0]);
00780
00781         if (m_pid == 0)
00782         {
00783             dup2(in[0], STDIN_FILENO);
00784             dup2(out[1], STDOUT_FILENO);
00785
00786             // Ignore stderr so that it doesnt pollute the console (e.g. GTK+ errors from zenity)
00787             int fd = open("/dev/null", O_WRONLY);
00788             dup2(fd, STDERR_FILENO);
00789             close(fd);
00790
00791             std::vector<char*> args;
00792             std::transform(command.cbegin(), command.cend(), std::back_inserter(args),
00793                 [](std::string const &s) { return const_cast<char*>(s.c_str()); });
00794             args.push_back(nullptr); // null-terminate argv[]
00795
00796             execvp(args[0], args.data());
00797             exit(1);
00798         }
00799
00800         close(in[1]);
00801         m_fd = out[0];
00802         auto flags = fcntl(m_fd, F_GETFL);
00803         fcntl(m_fd, F_SETFL, flags | O_NONBLOCK);
00804
00805         m_running = true;
00806     }
00807 #endif
00808
00809     inline internal::executor::~executor()
00810     {
00811         stop();
00812     }
00813
00814     inline bool internal::executor::ready(int timeout /* = default_wait_timeout */)
00815     {
00816         if (!m_running)
00817             return true;
00818
00819 #if _WIN32
00820         if (m_future.valid())
00821         {
00822             auto status = m_future.wait_for(std::chrono::milliseconds(timeout));
00823             if (status != std::future_status::ready)
00824             {
00825                 // On Windows, we need to run the message pump. If the async
00826                 // thread uses a Windows API dialog, it may be attached to the
00827                 // main thread and waiting for messages that only we can dispatch.
00828                 MSG msg;
00829                 while (PeekMessage(&msg, nullptr, 0, 0, PM_REMOVE))

```

```

00830         {
00831             TranslateMessage(&msg);
00832             DispatchMessage(&msg);
00833         }
00834         return false;
00835     }
00836
00837     m_stdout = m_future.get();
00838 }
00839 #elif __EMSCRIPTEN__ || __NX__
00840     // FIXME: do something
00841     (void)timeout;
00842 #else
00843     char buf[BUFSIZ];
00844     ssize_t received = read(m_fd, buf, BUFSIZ); // Flawfinder: ignore
00845     if (received > 0)
00846     {
00847         m_stdout += std::string(buf, received);
00848         return false;
00849     }
00850
00851     // Reap child process if it is dead. It is possible that the system has already reaped it
00852     // (this happens when the calling application handles or ignores SIG_CHLD) and results in
00853     // waitpid() failing with ECHILD. Otherwise we assume the child is running and we sleep for
00854     // a little while.
00855     int status;
00856     pid_t child = waitpid(m_pid, &status, WNOHANG);
00857     if (child != m_pid && (child >= 0 || errno != ECHILD))
00858     {
00859         // FIXME: this happens almost always at first iteration
00860         std::this_thread::sleep_for(std::chrono::milliseconds(timeout));
00861         return false;
00862     }
00863
00864     close(m_fd);
00865     m_exit_code = WEXITSTATUS(status);
00866 #endif
00867
00868     m_running = false;
00869     return true;
00870 }
00871
00872 inline void internal::executor::stop()
00873 {
00874     // Loop until the user closes the dialog
00875     while (!ready())
00876         ;
00877 }
00878
00879 // dll implementation
00880
00881 #if _WIN32
00882     inline internal::platform::dll::dll(std::string const &name)
00883     : handle(::LoadLibraryA(name.c_str()))
00884     {}
00885
00886     inline internal::platform::dll::~dll()
00887     {
00888         if (handle)
00889             ::FreeLibrary(handle);
00890     }
00891 #endif // _WIN32
00892
00893 // ole32_dll implementation
00894
00895 #if _WIN32
00896     inline internal::platform::ole32_dll::ole32_dll()
00897     : dll("ole32.dll")
00898     {
00899         // Use COINIT_MULTITHREADED because COINIT_APARTMENTTHREADED causes crashes.
00900         // See https://github.com/samhoevar/portable-file-dialogs/issues/51
00901         auto coinit = proc<HRESULT WINAPI (LPVOID, DWORD)>(&this, "CoInitializeEx");
00902         m_state = coinit(nullptr, COINIT_MULTITHREADED);
00903     }
00904
00905     inline internal::platform::ole32_dll::~ole32_dll()
00906     {
00907         if (is_initialized())
00908             proc<void WINAPI ()>(&this, "CoUninitialize")();
00909     }
00910
00911     inline bool internal::platform::ole32_dll::is_initialized()
00912     {
00913         return m_state == S_OK || m_state == S_FALSE;
00914     }
00915 #endif
00916

```



```

00917 // new_style_context implementation
00918
00919 #if _WIN32
00920     inline internal::platform::new_style_context::new_style_context()
00921     {
00922         // Only create one activation context for the whole app lifetime.
00923         static HANDLE hctx = create();
00924
00925         if (hctx != INVALID_HANDLE_VALUE)
00926             ActivateActCtx(hctx, &m_cookie);
00927     }
00928
00929     inline internal::platform::new_style_context::~new_style_context()
00930     {
00931         DeactivateActCtx(0, m_cookie);
00932     }
00933
00934     inline HANDLE internal::platform::new_style_context::create()
00935     {
00936         // This hack seems to be necessary for this code to work on windows XP.
00937         // Without it, dialogs do not show and close immediately. GetLastError()
00938         // returns 0 so I dont know what causes this. I was not able to reproduce
00939         // this behavior on Windows 7 and 10 but just in case, let it be here for
00940         // those versions too.
00941         // This hack is not required if other dialogs are used (they load comdlg32
00942         // automatically), only if message boxes are used.
00943         dll comdlg32("comdlg32.dll");
00944
00945         // Using approach as shown here: https://stackoverflow.com/a/10444161
00946         UINT len = ::GetSystemDirectoryA(nullptr, 0);
00947         std::string sys_dir(len, '\\0');
00948         ::GetSystemDirectoryA(&sys_dir[0], len);
00949
00950         ACTCTXA act_ctx =
00951         {
00952             // Do not set flag ACTCTX_FLAG_SET_PROCESS_DEFAULT, since it causes a
00953             // crash with error default context is already set.
00954             sizeof(act_ctx),
00955             ACTCTX_FLAG_RESOURCE_NAME_VALID | ACTCTX_FLAG_ASSEMBLY_DIRECTORY_VALID,
00956             "shell32.dll", 0, 0, sys_dir.c_str(), (LPCSTR)124, nullptr, 0,
00957         };
00958
00959         return ::CreateActCtxA(&act_ctx);
00960     }
00961 #endif // _WIN32
00962
00963 // dialog implementation
00964
00965     inline bool internal::dialog::ready(int timeout /* = default_wait_timeout */) const
00966     {
00967         return m_async->ready(timeout);
00968     }
00969
00970     inline bool internal::dialog::kill() const
00971     {
00972         return m_async->kill();
00973     }
00974
00975     inline internal::dialog::dialog()
00976         : m_async(std::make_shared<executor>())
00977     {
00978     }
00979
00980     inline std::vector<std::string> internal::dialog::desktop_helper() const
00981     {
00982         #if __APPLE__
00983             return { "osascript" };
00984         #else
00985             return { flags(flag::has_zenity) ? "zenity"
00986                     : flags(flag::has_matedialog) ? "matedialog"
00987                     : flags(flag::has_qarma) ?
00988                         "qarma"
00989                     : flags(flag::has_kdialog) ? "kdialog"
00990                     : "echo" };
00991         #endif
00992
00993     inline std::string internal::dialog::buttons_to_name(choice _choice)
00994     {
00995         switch (_choice)
00996         {
00997             case choice::ok_cancel: return "okcancel";
00998             case choice::yes_no: return "yesno";
00999             case choice::yes_no_cancel: return "yesnocancel";
01000             case choice::retry_cancel: return "retrycancel";

```

```

01001         case choice::abort_retry_ignore: return "abortretryignore";
01002         /* case choice::ok: */ default: return "ok";
01003     }
01004 }
01005
01006 inline std::string internal::dialog::get_icon_name(icon _icon)
01007 {
01008     switch (_icon)
01009     {
01010         case icon::warning: return "warning";
01011         case icon::error: return "error";
01012         case icon::question: return "question";
01013         // Zenity wants "information" but WinForms wants "info"
01014         /* case icon::info: */ default:
01015 #if _WIN32
01016             return "info";
01017 #else
01018             return "information";
01019 #endif
01020     }
01021 }
01022
01023 // This is only used for debugging purposes
01024 inline std::ostream& operator <(std::ostream &s, std::vector<std::string> const &v)
01025 {
01026     int not_first = 0;
01027     for (auto &e : v)
01028         s << (not_first++ ? " " : "") << e;
01029     return s;
01030 }
01031
01032 // Properly quote a string for Powershell: replace ' or " with " or ""
01033 // FIXME: we should probably get rid of newlines!
01034 // FIXME: the \ sequence seems unsafe, too!
01035 // XXX: this is no longer used but I would like to keep it around just in case
01036 inline std::string internal::dialog::powershell_quote(std::string const &str) const
01037 {
01038     return "\"" + std::regex_replace(str, std::regex("['\\""], "$&$&") + "\"";
01039 }
01040
01041 // Properly quote a string for osascript: replace \ or " with \\ or \"
01042 // XXX: this also used to replace ' with \' when popen was used, but it would be
01043 // smarter to do shell_quote(osascript_quote(...)) if this is needed again.
01044 inline std::string internal::dialog::osascript_quote(std::string const &str) const
01045 {
01046     return "\"" + std::regex_replace(str, std::regex("[\\\\\\\""]), "\\$&") + "\"";
01047 }
01048
01049 // Properly quote a string for the shell: just replace ' with '\"
01050 // XXX: this is no longer used but I would like to keep it around just in case
01051 inline std::string internal::dialog::shell_quote(std::string const &str) const
01052 {
01053     return "\"" + std::regex_replace(str, std::regex("'"), "\"\\\"") + "\"";
01054 }
01055
01056 // file_dialog implementation
01057
01058 inline internal::file_dialog::file_dialog(type in_type,
01059     std::string const &title,
01060     std::string const &default_path /* = "" */,
01061     std::vector<std::string> const &filters /* = {} */,
01062     opt options /* = opt::none */)
01063 {
01064 #if _WIN32
01065     std::string filter_list;
01066     std::regex whitespace(" ");
01067     for (size_t i = 0; i + 1 < filters.size(); i += 2)
01068     {
01069         filter_list += filters[i] + '\\0';
01070         filter_list += std::regex_replace(filters[i + 1], whitespace, ";") + '\\0';
01071     }
01072     filter_list += '\\0';
01073
01074     m_async->start_func([this, in_type, title, default_path, filter_list,
01075         options](int *exit_code) -> std::string
01076     {
01077         (void)exit_code;
01078         m_wtitle = internal::str2wstr(title);
01079         m_wdefault_path = internal::str2wstr(default_path);
01080         auto wfilter_list = internal::str2wstr(filter_list);
01081
01082         // Initialise COM. This is required for the new folder selection window,
01083         // (see https://github.com/samhocevar/portable-file-dialogs/pull/21)
01084         // and to avoid random crashes with GetOpenFileNameW() (see
01085         // https://github.com/samhocevar/portable-file-dialogs/issues/51)
01086         ole32_dll ole32;
01087

```

```

01088         // Folder selection uses a different method
01089         if (in_type == type::folder)
01090         {
01091             #if PFD_HAS_IFILEDIALOG
01092                 if (flags(flag::is_vista))
01093                 {
01094                     // On Vista and higher we should be able to use IFileDialog for folder selection
01095                     IFileDialog *ifd;
01096                     HRESULT hr = dll::proc<HRESULT WINAPI (REFCLSID, LPUNKNOWN, DWORD, REFIID, LPVOID
*)>(ole32, "CoCreateInstance")
01097                                     (CLSID_FileOpenDialog, nullptr, CLSCTX_INPROC_SERVER,
IID_PPV_ARGS(&ifd));
01098
01099                     // In case CoCreateInstance fails (which it should not), try legacy approach
01100                     if (SUCCEEDED(hr))
01101                         return select_folder_vista(ifd, options & opt::force_path);
01102                 }
01103             #endif
01104
01105             BROWSEINFOW bi;
01106             memset(&bi, 0, sizeof(bi));
01107
01108             bi.lpfn = &bffcallback;
01109             bi.lParam = (LPARAM)this;
01110
01111             if (flags(flag::is_vista))
01112             {
01113                 if (ole32.is_initialized())
01114                     bi.ulFlags |= BIF_NEWDIALOGSTYLE;
01115                 bi.ulFlags |= BIF_EDITBOX;
01116                 bi.ulFlags |= BIF_STATUSTEXT;
01117             }
01118
01119             auto *list = SHBrowseForFolderW(&bi);
01120             std::string ret;
01121             if (list)
01122             {
01123                 auto buffer = new wchar_t[MAX_PATH];
01124                 SHGetPathFromIDListW(list, buffer);
01125                 dll::proc<void WINAPI (LPVOID)>(ole32, "CoTaskMemFree")(list);
01126                 ret = internal::wstr2str(buffer);
01127                 delete[] buffer;
01128             }
01129             return ret;
01130         }
01131
01132         OPENFILENAMEW ofn;
01133         memset(&ofn, 0, sizeof(ofn));
01134         ofn.lStructSize = sizeof(OPENFILENAMEW);
01135         ofn.hwndOwner = GetActiveWindow();
01136
01137         ofn.lpstrFilter = wfilter_list.c_str();
01138
01139         auto woutput = std::wstring(MAX_PATH * 256, L'\0');
01140         ofn.lpstrFile = (LPWSTR)woutput.data();
01141         ofn.nMaxFile = (DWORD)woutput.size();
01142         if (!m_wdefault_path.empty())
01143         {
01144             // If a directory was provided, use it as the initial directory. If
01145             // a valid path was provided, use it as the initial file. Otherwise,
01146             // let the Windows API decide.
01147             auto path_attr = GetFileAttributesW(m_wdefault_path.c_str());
01148             if (path_attr != INVALID_FILE_ATTRIBUTES && (path_attr & FILE_ATTRIBUTE_DIRECTORY))
01149                 ofn.lpstrInitialDir = m_wdefault_path.c_str();
01150             else if (m_wdefault_path.size() <= woutput.size())
01151                 //second argument is size of buffer, not length of string
01152                 StringCchCopyW(ofn.lpstrFile, MAX_PATH*256+1, m_wdefault_path.c_str());
01153             else
01154             {
01155                 ofn.lpstrFileTitle = (LPWSTR)m_wdefault_path.data();
01156                 ofn.nMaxFileTitle = (DWORD)m_wdefault_path.size();
01157             }
01158         }
01159         ofn.lpstrTitle = m_wtitle.c_str();
01160         ofn.Flags = OFN_NOCHANGEDIR | OFN_EXPLORER;
01161
01162         dll comdlg32("comdlg32.dll");
01163
01164         // Apply new visual style (required for windows XP)
01165         new_style_context ctx;
01166
01167         if (in_type == type::save)
01168         {
01169             if (!(options & opt::force_overwrite))
01170                 ofn.Flags |= OFN_OVERWRITEPROMPT;
01171
01172             dll::proc<BOOL WINAPI (LPOPENFILENAMEW)> get_save_file_name(comdlg32, "GetSaveFileNameW");

```

```

01173         if (get_save_file_name(&ofn) == 0)
01174             return "";
01175         return internal::wstr2str(woutput.c_str());
01176     }
01177     else
01178     {
01179         if (options & opt::multiselect)
01180             ofn.Flags |= OFN_ALLOWMULTISELECT;
01181         ofn.Flags |= OFN_PATHMUSTEXIST;
01182
01183         dll::proc<BOOL WINAPI (LPOpenFileNameW)> get_open_file_name(comdlg32, "GetOpenFileNameW");
01184         if (get_open_file_name(&ofn) == 0)
01185             return "";
01186     }
01187
01188     std::string prefix;
01189     for (wchar_t const *p = woutput.c_str(); *p; )
01190     {
01191         auto filename = internal::wstr2str(p);
01192         p += wcslen(p);
01193         // In multiselect mode, we advance p one wchar further and
01194         // check for another filename. If there is one and the
01195         // prefix is empty, it means we just read the prefix.
01196         if ((options & opt::multiselect) && *++p && prefix.empty())
01197         {
01198             prefix = filename + "/";
01199             continue;
01200         }
01201
01202         m_vector_result.push_back(prefix + filename);
01203     }
01204
01205     return "";
01206 });
01207 #elif __EMSCRIPTEN__
01208     // FIXME: do something
01209     (void)in_type;
01210     (void)title;
01211     (void)default_path;
01212     (void)filters;
01213     (void)options;
01214 #else
01215     auto command = desktop_helper();
01216
01217     if (is_osascript())
01218     {
01219         std::string script = "set ret to choose";
01220         switch (in_type)
01221         {
01222             case type::save:
01223                 script += " file name";
01224                 break;
01225             case type::open: default:
01226                 script += " file";
01227                 if (options & opt::multiselect)
01228                     script += " with multiple selections allowed";
01229                 break;
01230             case type::folder:
01231                 script += " folder";
01232                 break;
01233         }
01234
01235         if (default_path.size())
01236         {
01237             if (in_type == type::folder || is_directory(default_path))
01238                 script += " default location ";
01239             else
01240                 script += " default name ";
01241             script += osascript_quote(default_path);
01242         }
01243
01244         script += " with prompt " + osascript_quote(title);
01245
01246         if (in_type == type::open)
01247         {
01248             // Concatenate all user-provided filter patterns
01249             std::string patterns;
01250             for (size_t i = 0; i < filters.size() / 2; ++i)
01251                 patterns += " " + filters[2 * i + 1];
01252
01253             // Split the pattern list to check whether "*" is in there; if it
01254             // is, we have to disable filters because there is no mechanism in
01255             // OS X for the user to override the filter.
01256             std::regex sep("\\s+");
01257             std::string filter_list;
01258             bool has_filter = true;
01259             std::sregex_token_iterator iter(patterns.begin(), patterns.end(), sep, -1);

```

```

01260         std::sregex_token_iterator end;
01261         for ( ; iter != end; ++iter)
01262         {
01263             auto pat = iter->str();
01264             if (pat == "*" || pat == ".*")
01265                 has_filter = false;
01266             else if (internal::starts_with(pat, "."))
01267                 filter_list += "," + osascript_quote(pat.substr(2, pat.size() - 2));
01268         }
01269
01270         if (has_filter && filter_list.size() > 0)
01271         {
01272             // There is a weird AppleScript bug where file extensions of length != 3 are
01273             // ignored, e.g. type{"txt"} works, but type{"json"} does not. Fortunately if
01274             // the whole list starts with a 3-character extension, everything works again.
01275             // We use "///" for such an extension because we are sure it cannot appear in
01276             // an actual filename.
01277             script += " of type {\\\"///\\\" + filter_list + \"}\";
01278         }
01279     }
01280
01281     if (in_type == type::open && (options & opt::multiselect))
01282     {
01283         script += "\nset s to \"\"";
01284         script += "\nrepeat with i in ret";
01285         script += "\n    set s to s & (POSIX path of i) & \"\\n\"";
01286         script += "\nend repeat";
01287         script += "\ncopy s to stdout";
01288     }
01289     else
01290     {
01291         script += "\nPOSIX path of ret";
01292     }
01293
01294     command.push_back("-e");
01295     command.push_back(script);
01296 }
01297 else if (is_zenity())
01298 {
01299     command.push_back("--file-selection");
01300
01301     // If the default path is a directory, make sure it ends with "/" otherwise zenity will
01302     // open the file dialog in the parent directory.
01303     auto filename_arg = "--filename=" + default_path;
01304     if (in_type != type::folder && !ends_with(default_path, "/") &&
        internal::is_directory(default_path))
01305         filename_arg += "/";
01306     command.push_back(filename_arg);
01307
01308     command.push_back("--title");
01309     command.push_back(title);
01310     command.push_back("--separator=\n");
01311
01312     for (size_t i = 0; i < filters.size() / 2; ++i)
01313     {
01314         command.push_back("--file-filter");
01315         command.push_back(filters[2 * i] + "|" + filters[2 * i + 1]);
01316     }
01317
01318     if (in_type == type::save)
01319         command.push_back("--save");
01320     if (in_type == type::folder)
01321         command.push_back("--directory");
01322     if (!(options & opt::force_overwrite))
01323         command.push_back("--confirm-overwrite");
01324     if (options & opt::multiselect)
01325         command.push_back("--multiple");
01326 }
01327 else if (is_kdialog())
01328 {
01329     switch (in_type)
01330     {
01331         case type::save: command.push_back("--getsavefilename"); break;
01332         case type::open: command.push_back("--getopenfilename"); break;
01333         case type::folder: command.push_back("--getexistingdirectory"); break;
01334     }
01335     if (options & opt::multiselect)
01336     {
01337         command.push_back("--multiple");
01338         command.push_back("--separate-output");
01339     }
01340
01341     command.push_back(default_path);
01342
01343     std::string filter;
01344     for (size_t i = 0; i < filters.size() / 2; ++i)
01345         filter += (i == 0 ? "" : " | ") + filters[2 * i] + "(" + filters[2 * i + 1] + ")";

```

```

01346         command.push_back(filter);
01347
01348         command.push_back("--title");
01349         command.push_back(title);
01350     }
01351
01352     if (flags(flag::is_verbose))
01353         std::cerr << "pfd: " << command << std::endl;
01354
01355     m_async->start_process(command);
01356 #endif
01357 }
01358
01359 inline std::string internal::file_dialog::string_result()
01360 {
01361 #if _WIN32
01362     return m_async->result();
01363 #else
01364     auto ret = m_async->result();
01365     // Strip potential trailing newline (zenity). Also strip trailing slash
01366     // added by osascript for consistency with other backends.
01367     while (!ret.empty() && (ret.back() == '\\n' || ret.back() == '/'))
01368         ret.pop_back();
01369     return ret;
01370 #endif
01371 }
01372
01373 inline std::vector<std::string> internal::file_dialog::vector_result()
01374 {
01375 #if _WIN32
01376     m_async->result();
01377     return m_vector_result;
01378 #else
01379     std::vector<std::string> ret;
01380     auto result = m_async->result();
01381     for (;;)
01382     {
01383         // Split result along newline characters
01384         auto i = result.find('\\n');
01385         if (i == 0 || i == std::string::npos)
01386             break;
01387         ret.push_back(result.substr(0, i));
01388         result = result.substr(i + 1, result.size());
01389     }
01390     return ret;
01391 #endif
01392 }
01393
01394 #if _WIN32
01395 // Use a static function to pass as BFFCALLBACK for legacy folder select
01396 inline int CALLBACK internal::file_dialog::bffcallback(HWND hwnd, UINT uMsg,
01397     LPARAM, LPARAM pData)
01398 {
01399     auto inst = (file_dialog *)pData;
01400     switch (uMsg)
01401     {
01402     case BFFM_INITIALIZED:
01403         SendMessage(hwnd, BFFM_SETSELECTIONW, TRUE, (LPARAM)inst->m_wdefault_path.c_str());
01404         break;
01405     }
01406     return 0;
01407 }
01408
01409 #if PFD_HAS_IFILEDIALOG
01410 inline std::string internal::file_dialog::select_folder_vista(IFileDialog *ifd, bool force_path)
01411 {
01412     std::string result;
01413
01414     IShellItem *folder;
01415
01416     // Load library at runtime so app doesn't link it at load time (which will fail on windows XP)
01417     dll shell32("shell32.dll");
01418     dll::proc<HRESULT WINAPI (PCWSTR, IBindCtx*, REFIID, void**)>
01419         create_item(shell32, "SHCreateItemFromParsingName");
01420
01421     if (!create_item)
01422         return "";
01423
01424     auto hr = create_item(m_wdefault_path.c_str(),
01425         nullptr,
01426         IID_PPV_ARGS(&folder));
01427
01428     // Set default folder if found. This only sets the default folder. If
01429     // Windows has any info about the most recently selected folder, it
01430     // will display it instead. Generally, calling SetFolder() to set the
01431     // current directory is not a good or expected user experience and
01432     // should therefore be avoided:

```

```

01433     //
01434     https://docs.microsoft.com/windows/win32/api/shobjidl_core/nf-shobjidl_core-ifiledialog-setfolder
01434     if (SUCCEEDED(hr))
01435     {
01436         if (force_path)
01437             ifd->SetFolder(folder);
01438         else
01439             ifd->SetDefaultFolder(folder);
01440         folder->Release();
01441     }
01442
01443     // Set the dialog title and option to select folders
01444     ifd->SetOptions(FOS_PICKFOLDERS | FOS_FORCEFILESYSTEM);
01445     ifd->SetTitle(m_wtitle.c_str());
01446
01447     hr = ifd->Show(GetActiveWindow());
01448     if (SUCCEEDED(hr))
01449     {
01450         IShellItem* item;
01451         hr = ifd->GetResult(&item);
01452         if (SUCCEEDED(hr))
01453         {
01454             wchar_t* wname = nullptr;
01455             // This is unlikely to fail because we use FOS_FORCEFILESYSTEM, but try
01456             // to output a debug message just in case.
01457             if (SUCCEEDED(item->GetDisplayName(SIGDN_FILESYSPATH, &wname)))
01458             {
01459                 result = internal::wstr2str(std::wstring(wname));
01460                 dll::proc<void WINAPI (LPVOID)>(ole32_dll(), "CoTaskMemFree")(wname);
01461             }
01462             else
01463             {
01464                 if (SUCCEEDED(item->GetDisplayName(SIGDN_NORMALDISPLAY, &wname)))
01465                 {
01466                     auto name = internal::wstr2str(std::wstring(wname));
01467                     dll::proc<void WINAPI (LPVOID)>(ole32_dll(), "CoTaskMemFree")(wname);
01468                     std::cerr << "pfd: failed to get path for " << name << std::endl;
01469                 }
01470                 else
01471                     std::cerr << "pfd: item of unknown type selected" << std::endl;
01472             }
01473             item->Release();
01474         }
01475     }
01476     ifd->Release();
01477     return result;
01478 }
01479 #endif
01480 #endif
01481
01482 // notify implementation
01483
01484 inline notify::notify(std::string const &title,
01485                       std::string const &message,
01486                       icon _icon /* = icon::info */)
01487 {
01488     if (_icon == icon::question) // Not supported by notifications
01489         _icon = icon::info;
01490
01491 #if _WIN32
01492     // Use a static shared pointer for notify_icon so that we can delete
01493     // it whenever we need to display a new one, and we can also wait
01494     // until the program has finished running.
01495     struct notify_icon_data : public NOTIFYICONDATAW
01496     {
01497         ~notify_icon_data() { Shell_NotifyIconW(NIM_DELETE, this); }
01498     };
01499
01500     static std::shared_ptr<notify_icon_data> nid;
01501
01502     // Release the previous notification icon, if any, and allocate a new
01503     // one. Note that std::make_shared() does value initialization, so there
01504     // is no need to memset the structure.
01505     nid = nullptr;
01506     nid = std::make_shared<notify_icon_data>();
01507
01508     // For XP support
01509     nid->cbSize = NOTIFYICONDATAW_V2_SIZE;
01510     nid->hWnd = nullptr;
01511     nid->uID = 0;
01512
01513     // Flag Description:
01514     // - NIF_ICON The hIcon member is valid.
01515     // - NIF_MESSAGE The uCallbackMessage member is valid.

```

```

01519 // - NIF_TIP      The szTip member is valid.
01520 // - NIF_STATE    The dwState and dwStateMask members are valid.
01521 // - NIF_INFO     Use a balloon ToolTip instead of a standard ToolTip. The szInfo, uTimeout,
szInfoTitle, and dwInfoFlags members are valid.
01522 // - NIF_GUID     Reserved.
01523 nid->uFlags = NIF_MESSAGE | NIF_ICON | NIF_INFO;
01524
01525 // Flag Description
01526 // - NIIF_ERROR    An error icon.
01527 // - NIIF_INFO     An information icon.
01528 // - NIIF_NONE     No icon.
01529 // - NIIF_WARNING  A warning icon.
01530 // - NIIF_ICON_MASK Version 6.0. Reserved.
01531 // - NIIF_NOSOUND  Version 6.0. Do not play the associated sound. Applies only to balloon
ToolTip
01532 switch (_icon)
01533 {
01534     case icon::warning: nid->dwInfoFlags = NIIF_WARNING; break;
01535     case icon::error: nid->dwInfoFlags = NIIF_ERROR; break;
01536     /* case icon::info: */ default: nid->dwInfoFlags = NIIF_INFO; break;
01537 }
01538
01539 ENUMRESNAMEPROC icon_enum_callback = [] (HMODULE, LPCTSTR, LPTSTR lpName, LONG_PTR lParam) -> BOOL
01540 {
01541     ((NOTIFYICONDATAW *)lParam)->hIcon = ::LoadIcon(GetModuleHandle(nullptr), lpName);
01542     return false;
01543 };
01544
01545 nid->hIcon = ::LoadIcon(nullptr, IDI_APPLICATION);
01546 ::EnumResourceNames(nullptr, RT_GROUP_ICON, icon_enum_callback, (LONG_PTR)nid.get());
01547
01548 nid->uTimeout = 5000;
01549
01550 StringCchCopyW(nid->szInfoTitle, ARRAYSIZE(nid->szInfoTitle), internal::str2wstr(title).c_str());
01551 StringCchCopyW(nid->szInfo, ARRAYSIZE(nid->szInfo), internal::str2wstr(message).c_str());
01552
01553 // Display the new icon
01554 Shell_NotifyIconW(NIM_ADD, nid.get());
01555 #elif __EMSCRIPTEN__
01556 // FIXME: do something
01557 (void)title;
01558 (void)message;
01559 #else
01560 auto command = desktop_helper();
01561
01562 if (is_osascript())
01563 {
01564     command.push_back("-e");
01565     command.push_back("display notification " + osascript_quote(message) +
" with title " + osascript_quote(title));
01566 }
01567 else if (is_zenity())
01568 {
01569     command.push_back("--notification");
01570     command.push_back("--window-icon");
01571     command.push_back(get_icon_name(_icon));
01572     command.push_back("--text");
01573     command.push_back(title + "\n" + message);
01574 }
01575 else if (is_kdialog())
01576 {
01577     command.push_back("--icon");
01578     command.push_back(get_icon_name(_icon));
01579     command.push_back("--title");
01580     command.push_back(title);
01581     command.push_back("--passivepopup");
01582     command.push_back(message);
01583     command.push_back("5");
01584 }
01585
01586 if (flags(flag::is_verbose))
01587     std::cerr << "pfd: " << command << std::endl;
01588
01589 m_async->start_process(command);
01590 #endif
01591 }
01592
01593 // message implementation
01594
01595 inline message::message(std::string const &title,
std::string const &text,
choice _choice /* = choice::ok_cancel */,
icon _icon /* = icon::info */)
01596 {
01597     {
01598         #if _WIN32
01599             // Use MB_SYSTEMMODAL rather than MB_TOPMOST to ensure the message window is brought
01600             // to front. See https://github.com/samhocevar/portable-file-dialogs/issues/52

```



```

01604     UINT style = MB_SYSTEMMODAL;
01605     switch (_icon)
01606     {
01607         case icon::warning: style |= MB_ICONWARNING; break;
01608         case icon::error: style |= MB_ICONERROR; break;
01609         case icon::question: style |= MB_ICONQUESTION; break;
01610         /* case icon::info: */ default: style |= MB_ICONINFORMATION; break;
01611     }
01612
01613     switch (_choice)
01614     {
01615         case choice::ok_cancel: style |= MB_OKCANCEL; break;
01616         case choice::yes_no: style |= MB_YESNO; break;
01617         case choice::yes_no_cancel: style |= MB_YESNOCANCEL; break;
01618         case choice::retry_cancel: style |= MB_RETRYCANCEL; break;
01619         case choice::abort_retry_ignore: style |= MB_ABORTRETRYIGNORE; break;
01620         /* case choice::ok: */ default: style |= MB_OK; break;
01621     }
01622
01623     m_mappings[IDCANCEL] = button::cancel;
01624     m_mappings[IDOK] = button::ok;
01625     m_mappings[IDYES] = button::yes;
01626     m_mappings[IDNO] = button::no;
01627     m_mappings[IDABORT] = button::abort;
01628     m_mappings[IDRETRY] = button::retry;
01629     m_mappings[IDIGNORE] = button::ignore;
01630
01631     m_async->start_func([text, title, style](int* exit_code) -> std::string
01632     {
01633         auto wtext = internal::str2wstr(text);
01634         auto wtitle = internal::str2wstr(title);
01635         // Apply new visual style (required for all Windows versions)
01636         new_style_context ctx;
01637         *exit_code = MessageBoxW(GetActiveWindow(), wtext.c_str(), wtitle.c_str(), style);
01638         return "";
01639     });
01640
01641 #elif __EMSCRIPTEN__
01642     std::string full_message;
01643     switch (_icon)
01644     {
01645         case icon::warning: full_message = ""; break;
01646         case icon::error: full_message = ""; break;
01647         case icon::question: full_message = ""; break;
01648         /* case icon::info: */ default: full_message = ""; break;
01649     }
01650
01651     full_message += ' ' + title + "\n\n" + text;
01652
01653     // This does not really start an async task; it just passes the
01654     // EM_ASM_INT return value to a fake start() function.
01655     m_async->start(EM_ASM_INT{
01656     {
01657         if ($1)
01658             return window.confirm(UTF8ToString($0)) ? 0 : -1;
01659         alert(UTF8ToString($0));
01660         return 0;
01661     }, full_message.c_str(), _choice == choice::ok_cancel));
01662 #else
01663     auto command = desktop_helper();
01664
01665     if (is_osascript())
01666     {
01667         std::string script = "display dialog " + osascript_quote(text) +
01668                             " with title " + osascript_quote(title);
01669         auto if_cancel = button::cancel;
01670         switch (_choice)
01671         {
01672             case choice::ok_cancel:
01673                 script += "buttons {\\"OK\\", \\"Cancel\\"}"
01674                         " default button \\"OK\\" "
01675                         " cancel button \\"Cancel\\"";
01676                 break;
01677             case choice::yes_no:
01678                 script += "buttons {\\"Yes\\", \\"No\\"}"
01679                         " default button \\"Yes\\" "
01680                         " cancel button \\"No\\"";
01681                 if_cancel = button::no;
01682                 break;
01683             case choice::yes_no_cancel:
01684                 script += "buttons {\\"Yes\\", \\"No\\", \\"Cancel\\"}"
01685                         " default button \\"Yes\\" "
01686                         " cancel button \\"Cancel\\"";
01687                 break;
01688             case choice::retry_cancel:
01689                 script += "buttons {\\"Retry\\", \\"Cancel\\"}"
01690                         " default button \\"Retry\\"";

```

```

01691         " cancel button \"Cancel\"";
01692     break;
01693     case choice::abort_retry_ignore:
01694         script += "buttons {\\"Abort\\", \\"Retry\\", \\"Ignore\\"}"
01695             " default button \\"Abort\\"";
01696             " cancel button \\"Retry\\"";
01697         if_cancel = button::retry;
01698     break;
01699     case choice::ok: default:
01700         script += "buttons {\\"OK\\"}"
01701             " default button \\"OK\\"";
01702             " cancel button \\"OK\\"";
01703         if_cancel = button::ok;
01704     break;
01705 }
01706 m_mappings[1] = if_cancel;
01707 m_mappings[256] = if_cancel; // XXX: I think this was never correct
01708 script += " with icon ";
01709 switch (_icon)
01710 {
01711 #define PFD_OSX_ICON(n) "alias ((path to library folder from system domain) as text " \
01712     "& \\"CoreServices:CoreTypes.bundle:Contents:Resources:" n ".icns\\")"
01713     case icon::info: default: script += PFD_OSX_ICON("ToolBarInfo"); break;
01714     case icon::warning: script += "caution"; break;
01715     case icon::error: script += "stop"; break;
01716     case icon::question: script += PFD_OSX_ICON("GenericQuestionMarkIcon"); break;
01717 #undef PFD_OSX_ICON
01718 }
01719
01720 command.push_back("-e");
01721 command.push_back(script);
01722 }
01723 else if (is_zenity())
01724 {
01725     switch (_choice)
01726     {
01727     case choice::ok_cancel:
01728         command.insert(command.end(), { "--question", "--cancel-label=Cancel",
01729             "--ok-label=OK" }); break;
01730     case choice::yes_no:
01731         // Do not use standard --question because it causes No to return -1,
01732         // which is inconsistent with the Yes/No/Cancel mode below.
01733         command.insert(command.end(), { "--question", "--switch", "--extra-button=No",
01734             "--extra-button=Yes" }); break;
01735     case choice::yes_no_cancel:
01736         command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
01737             "--extra-button=No", "--extra-button=Yes" }); break;
01738     case choice::retry_cancel:
01739         command.insert(command.end(), { "--question", "--switch", "--extra-button=Cancel",
01740             "--extra-button=Retry" }); break;
01741     case choice::abort_retry_ignore:
01742         command.insert(command.end(), { "--question", "--switch", "--extra-button=Ignore",
01743             "--extra-button=Abort", "--extra-button=Retry" }); break;
01744     case choice::ok:
01745     default:
01746         switch (_icon)
01747         {
01748             case icon::error: command.push_back("--error"); break;
01749             case icon::warning: command.push_back("--warning"); break;
01750             default: command.push_back("--info"); break;
01751         }
01752     }
01753     command.insert(command.end(), { "--title", title,
01754         "--width=300", "--height=0", // sensible defaults
01755         "--no-markup", // do not interpret text as Pango markup
01756         "--text", text,
01757         "--icon-name=dialog-" + get_icon_name(_icon) });
01758 }
01759 else if (is_kdialog())
01760 {
01761     if (_choice == choice::ok)
01762     {
01763         switch (_icon)
01764         {
01765             case icon::error: command.push_back("--error"); break;
01766             case icon::warning: command.push_back("--sorry"); break;
01767             default: command.push_back("--msgbox"); break;
01768         }
01769     }
01770     else
01771     {
01772         std::string flag = "--";
01773         if (_icon == icon::warning || _icon == icon::error)
01774             flag += "warning";
01775         flag += "yesno";
01776         if (_choice == choice::yes_no_cancel)

```

```

01773         flag += "cancel";
01774         command.push_back(flag);
01775         if (_choice == choice::yes_no || _choice == choice::yes_no_cancel)
01776         {
01777             m_mappings[0] = button::yes;
01778             m_mappings[256] = button::no;
01779         }
01780     }
01781
01782     command.push_back(text);
01783     command.push_back("--title");
01784     command.push_back(title);
01785
01786     // Must be after the above part
01787     if (_choice == choice::ok_cancel)
01788         command.insert(command.end(), { "--yes-label", "OK", "--no-label", "Cancel" });
01789 }
01790
01791 if (flags(flag::is_verbose))
01792     std::cerr << "pfd: " << command << std::endl;
01793
01794 m_async->start_process(command);
01795 #endif
01796 }
01797
01798 inline button message::result()
01799 {
01800     int exit_code;
01801     auto ret = m_async->result(&exit_code);
01802     // osascript will say "button returned:Cancel\n"
01803     // and others will just say "Cancel\n"
01804     if (internal::ends_with(ret, "Cancel\n"))
01805         return button::cancel;
01806     if (internal::ends_with(ret, "OK\n"))
01807         return button::ok;
01808     if (internal::ends_with(ret, "Yes\n"))
01809         return button::yes;
01810     if (internal::ends_with(ret, "No\n"))
01811         return button::no;
01812     if (internal::ends_with(ret, "Abort\n"))
01813         return button::abort;
01814     if (internal::ends_with(ret, "Retry\n"))
01815         return button::retry;
01816     if (internal::ends_with(ret, "Ignore\n"))
01817         return button::ignore;
01818     if (m_mappings.count(exit_code) != 0)
01819         return m_mappings[exit_code];
01820     return exit_code == 0 ? button::ok : button::cancel;
01821 }
01822
01823 // open_file implementation
01824
01825 inline open_file::open_file(std::string const &title,
01826                             std::string const &default_path /* = "" */,
01827                             std::vector<std::string> const &filters /* = { "All Files", "*" } */,
01828                             opt options /* = opt::none */)
01829     : file_dialog(type::open, title, default_path, filters, options)
01830 {
01831 }
01832
01833 inline open_file::open_file(std::string const &title,
01834                             std::string const &default_path,
01835                             std::vector<std::string> const &filters,
01836                             bool allow_multiselect)
01837     : open_file(title, default_path, filters,
01838                 (allow_multiselect ? opt::multiselect : opt::none))
01839 {
01840 }
01841
01842 inline std::vector<std::string> open_file::result()
01843 {
01844     return vector_result();
01845 }
01846
01847 // save_file implementation
01848
01849 inline save_file::save_file(std::string const &title,
01850                             std::string const &default_path /* = "" */,
01851                             std::vector<std::string> const &filters /* = { "All Files", "*" } */,
01852                             opt options /* = opt::none */)
01853     : file_dialog(type::save, title, default_path, filters, options)
01854 {
01855 }
01856
01857 inline save_file::save_file(std::string const &title,
01858                             std::string const &default_path,
01859                             std::vector<std::string> const &filters,

```

```

01860                                     bool confirm_overwrite)
01861     : save_file(title, default_path, filters,
01862               (confirm_overwrite ? opt::none : opt::force_overwrite))
01863     {
01864     }
01865
01866     inline std::string save_file::result()
01867     {
01868         return string_result();
01869     }
01870
01871 // select_folder implementation
01872
01873     inline select_folder::select_folder(std::string const &title,
01874                                         std::string const &default_path /* = "" */ ,
01875                                         opt options /* = opt::none */)
01876     : file_dialog(type::folder, title, default_path, {}, options)
01877     {
01878     }
01879
01880     inline std::string select_folder::result()
01881     {
01882         return string_result();
01883     }
01884
01885 #endif // PFD_SKIP_IMPLEMENTATION
01886
01887 } // namespace pfd

```

## 8.13 include/core/LinkedList.cpp File Reference

```
#include "LinkedList.hpp"
```

## 8.14 LinkedList.cpp

[Go to the documentation of this file.](#)

```

00001 //
00002 // Created by dirii on 12/04/2023.
00003 //
00004
00005 #include "LinkedList.hpp"
00006
00007 LinkedList::LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList) {
00008     this->window = window;
00009     this->typeLinkedList = typeLinkedList;
00010     this->highlighter = nullptr;
00011     this->delayTime = constants::LinkedList::DELAY_TIME;
00012     this->backArrow = new BackArrow(this->window, {0, 0}, {0, 0});
00013
00014     if (this->typeLinkedList == TypeLinkedList::CIRCULAR)
00015         this->backArrow->show();
00016     else
00017         this->backArrow->hide();
00018
00019     this->createLinkedList(0);
00020 }
00021
00022 void LinkedList::clear() {
00023     for (auto &node : this->nodes)
00024         delete node;
00025     this->nodes.clear();
00026     this->size = 0;
00027 }
00028
00029 void LinkedList::render() {
00030     if (this->size > 1) {
00031         // this->backArrow->toggleActiveColorNode();
00032         this->backArrow->render();
00033     }
00034     for (auto &node : this->nodes) {
00035         node->render();
00036     }
00037 }
00038

```

```

00039 LinkedList::LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList, int size) {
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 }
00053
00054 LinkedList::LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList,
std::vector<std::string> values) {
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 }
00068
00069 void LinkedList::update() {
00070     if ((int)this->events.size() && (this->isDelay or this->clock.getElapsedTime().asSeconds() >
this->delayTime / this->speed))
00071         this->updateAnimation();
00072     this->isDelay = false;
00073 }
00074
00075 void LinkedList::updateAnimation() {
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     }
00146
00147     void LinkedList::calculateEffectivePositions() {
00148         if (this->size < 2) return;
00149
00150         int lastInChain = 0;
00151         if (this->nodes[lastInChain]->getStatusNode() != NodeInfo::StatusNode::InChain) {
00152             lastInChain++;
00153         }
00154
00155         this->nodes[lastInChain]->setEffectivePosition(
00156             sf::Vector2f(
00157                 constants::NodeInfo::originNode.x,
00158                 constants::NodeInfo::originNode.y
00159             )
00160         );
00161
00162         for (int i = lastInChain + 1; i < this->size; i++){
00163             if (this->nodes[i]->getStatusNode() == NodeInfo::StatusNode::InChain) {
00164                 this->nodes[i]->setEffectivePosition(
00165                     sf::Vector2f(
00166                         this->nodes[lastInChain]->getPosition().x + constants::NodeInfo::offsetX,
00167                         this->nodes[lastInChain]->getPosition().y
00168                     )
00169                 );
00170                 lastInChain = i;
00171             }
00172         }
00173     }
00174
00175     void LinkedList::resetEvents() {
00176         delete this->highlighter;
00177         this->highlighter = nullptr;
00178         this->currentEvent = 0;
00179         this->events.clear();
00180         this->chosenNode = 0;
00181
00182         if (this->deletedNode != -1){
00183             this->nodes.erase(this->nodes.begin() + this->deletedNode);
00184             --this->size;
00185             if (this->size && this->deletedNode == this->size)
00186                 this->nodes.back()->destroyArrow(NodeInfo::ArrowType::RIGHT);
00187             if (this->size && this->deletedNode == 0)
00188                 this->nodes[0]->destroyArrow(NodeInfo::ArrowType::LEFT);
00189         }
00190         this->deletedNode = -1;
00191
00192         for (int i = 0; i < this->size; i++){
00193             this->nodes[i]->reset();
00194             this->nodes[i]->reInitPos(i);
00195             this->nodes[i]->reInitPreVal();
00196         }
00197         if (this->size > 1)
00198             this->backArrow->setPosition(this->nodes.back()->getPosition(),
this->nodes[0]->getPosition());
00199     }
00200
00201     void LinkedList::createLinkedList(int _size) {
00202         this->resetEvents();
00203         this->size = _size;
00204         for (auto &node : this->nodes)
00205             delete node;

```

```

00206     this->nodes.resize(_size);
00207     for (int i = 0; i < size; i++){
00208         this->nodes[i] = new NodeInfo(
00209             this->window,
00210             std::to_string(Random::randomInt(0, 99)),
00211             sf::Vector2f(
00212                 constants::NodeInfo::originNode.x + static_cast<float>(i) *
constants::NodeInfo::offsetX,
00213                 constants::NodeInfo::originNode.y
00214             ),
00215             this->typeLinkedList == TypeLinkedList::DOUBLY
00216         );
00217         if (i > 0){
00218             this->nodes[i - 1]->initArrow(
00219                 NodeInfo::ArrowType::RIGHT,
00220                 this->nodes[i - 1]->getPosition(),
00221                 this->nodes[i]->getPosition()
00222             );
00223             if (this->typeLinkedList == TypeLinkedList::DOUBLY)
00224                 this->nodes[i]->initArrow(
00225                     NodeInfo::ArrowType::LEFT,
00226                     this->nodes[i]->getPosition(),
00227                     this->nodes[i - 1]->getPosition()
00228                 );
00229         }
00230     }
00231     if (this->size > 1)
00232         this->backArrow->setPosition(this->nodes.back()->getPosition(),
this->nodes[0]->getPosition());
00233 }
00234
00235 void LinkedList::createLinkedList(std::vector<std::string> values) {
00236     this->resetEvents();
00237     this->size = static_cast<int>(values.size());
00238     for (auto &node : this->nodes)
00239         delete node;
00240     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 }
00268
00269 void LinkedList::initHighlighter(int linesCount, const char *codePath) {
00270     delete this->highlighter;
00271     this->highlighter = new Highlighter(
00272         this->window,
00273         linesCount,
00274         codePath
00275     );
00276 }
00277
00278 void LinkedList::toggleLines(std::vector<int> lines) {
00279     this->highlighter->toggle(std::move(lines));
00280 }
00281
00282 void LinkedList::renderHighlighter() {
00283     if (this->highlighter)
00284         this->highlighter->render();
00285 }
00286
00287 void LinkedList::processControlMenu(ControlMenu::StatusCode status) {
00288     if (this->clock.getElapsedTime().asSeconds() < this->delayTime / this->speed)

```

```

00289         return;
00290     switch (status){
00291     case ControlMenu::StatusCode::PREVIOUS:
00292         if (this->currentEvent > 0)
00293             --this->currentEvent;
00294         break;
00295     case ControlMenu::StatusCode::PAUSE:
00296         // std::cout << "PAUSE" << std::endl;
00297         break;
00298     case ControlMenu::StatusCode::PLAY:
00299         if (this->currentEvent + 1 < this->events.size()) {
00300             this->isDelay = true;
00301             this->clock.restart();
00302         }
00303     case ControlMenu::StatusCode::NEXT:
00304         if (this->currentEvent + 1 < this->events.size())
00305             ++this->currentEvent;
00306         break;
00307     default:
00308         break;
00309     }
00310 }
00311
00312 void LinkedList::setSpeed(float _speed) {
00313     this->speed = _speed;
00314 }
00315
00316 int LinkedList::getSize() const {
00317     return this->size;
00318 }
00319
00320 void LinkedList::addNode(int position, std::string value, const std::vector<EventAnimation>&
listEvents) {
00321     if (position < 0 || position > this->size) return;
00322
00323     sf::Vector2f newPosition(
00324         constants::NodeInfo::originNode.x + static_cast<float>(this->nodes.size()) *
constants::NodeInfo::offsetX,
00325         constants::NodeInfo::originNode.y
00326     );
00327     if (this->size) {
00328         this->nodes.back()->initArrow(
00329             NodeInfo::ArrowType::RIGHT,
00330             this->nodes.back()->getPosition(),
00331             newPosition
00332         );
00333     }
00334     this->nodes.push_back(new NodeInfo(
00335         this->window,
00336         "10",
00337         newPosition,
00338         this->typeLinkedList == TypeLinkedList::DOUBLY
00339     ));
00340     ++this->size;
00341     if (this->typeLinkedList == TypeLinkedList::DOUBLY && this->size > 1)
00342         this->nodes.back()->initArrow(
00343             NodeInfo::ArrowType::LEFT,
00344             this->nodes.back()->getPosition(),
00345             this->nodes[this->nodes.size() - 2]->getPosition()
00346         );
00347     this->backArrow->setPosition(newPosition, this->nodes[0]->getPosition());
00348     for (int i = this->size - 1; i > position; --i) {
00349         this->nodes[i]->setValue(this->nodes[i - 1]->getValue());
00350         this->nodes[i]->reInitPreVal();
00351     }
00352     this->nodes[position]->setValue(std::move(value));
00353     // std::cout << "add node to the current list " << position << " " << this->nodes[position]->getValue()
<< std::endl;
00354
00355     this->chosenNode = position;
00356     this->currentEvent = 0;
00357
00358     for (auto &e : listEvents)
00359         this->events.emplace_back(e);
00360 }
00361
00362 void LinkedList::deleteNode(int position, const std::vector<EventAnimation>& listEvents) {
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 }
00372

```



```

00373 void LinkedList::updateNode(int position, std::string value, const std::vector<EventAnimation>
    &listEvents) {
00374     if (position < 0 || position >= this->size) return;
00375
00376     this->nodes[position]->setValue(std::move(value));
00377     this->chosenNode = position;
00378     this->currentEvent = 0;
00379
00380     for (auto &e : listEvents)
00381         this->events.emplace_back(e);
00382 }
00383
00384 void LinkedList::searchNode(const std::vector<EventAnimation> &listEvents) {
00385     this->chosenNode = 0;
00386     this->currentEvent = 0;
00387
00388     for (auto &e : listEvents)
00389         this->events.emplace_back(e);
00390 }
00391
00392 int LinkedList::findValue(const std::string& value) {
00393     for (int i = 0; i < this->size; ++i)
00394         if (this->nodes[i]->getValue() == value)
00395             return i;
00396     return this->size;
00397 }
00398
00399 sf::Vector2f LinkedList::getPosNode(int position) {
00400     if (position < 0 || position >= this->size) return {};
00401     return this->nodes[position]->getPosition();
00402 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 12/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_LINKEDLIST_HPP
00006 #define VISUALGO_CS162_LINKEDLIST_HPP
00007
00008 #include "Random.h"
00009 #include "draw/NodeInfo.hpp"
00010 #include "draw/BackArrow.hpp"
00011 #include "libScene/Highlighter.hpp"
00012 #include "libScene/ControlMenu.hpp"
00013 #include "EventAnimation.hpp"
00014 #include "core/Vector.h"
00015
00016 class LinkedList {
00017 public:
00018     enum class TypeLinkedList{

```

```

00019         SINGLY,
00020         DOUBLY,
00021         CIRCULAR
00022     };
00023
00024     explicit LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList);
00025     LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList, int size);
00026     LinkedList(sf::RenderWindow* window, TypeLinkedList typeLinkedList, std::vector<std::string>
values);
00027
00028     void setSpeed(float speed);
00029     int findValue(const std::string& value);
00030     sf::Vector2f getPosNode(int position);
00031
00032     [[nodiscard]] int getSize() const;
00033
00034     void update();
00035     void updateAnimation();
00036     void render();
00037     void renderHighlighter();
00038     void resetEvents();
00039
00040     void calculateEffectivePositions();
00041     void clear();
00042
00043     void processControlMenu(ControlMenu::StatusCode status);
00044
00045     // operations of highlighter
00046     void initHighlighter(int linesCount, const char *codePath);
00047     void toggleLines(std::vector<int> lines);
00048
00049     // operations of linked list
00050     void createLinkedList(int size);
00051     void createLinkedList(std::vector<std::string> values);
00052     void addNode(int position, std::string value, const std::vector<EventAnimation>& listEvents);
00053     void deleteNode(int position, const std::vector<EventAnimation>& listEvents);
00054     void updateNode(int position, std::string value, const std::vector<EventAnimation>& listEvents);
00055     void searchNode(const std::vector<EventAnimation>& listEvents);
00056
00057 private:
00058     sf::RenderWindow* window;
00059     sf::Clock clock;
00060     int chosenNode = 0, deletedNode = -1;
00061     TypeLinkedList typeLinkedList;
00062
00063     Vector<NodeInfo*> nodes;
00064     int size;
00065
00066     BackArrow* backArrow;
00067
00068     Highlighter* highlighter;
00069
00070     std::vector<EventAnimation> events;
00071     int currentEvent;
00072
00073     float speed, delayTime;
00074     bool isDelay = false;
00075 };
00076
00077 #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.](#)

```
00001 //
00002 // Created by dirii on 01/05/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_RANDOM_H
00006 #define VISUALGO_CS162_RANDOM_H
00007
00008 #include <random>
00009
00010 namespace Random{
00011     static std::mt19937 rng(std::random_device{}());
00012     static int randomInt(int min, int max){
00013         std::uniform_int_distribution<int> dist(min, max);
00014         return dist(rng);
00015     }
00016 }
00017
00018 #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.](#)

```
00001 //
00002 // Created by dirii on 27/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_VECTOR_H
00006 #define VISUALGO_CS162_VECTOR_H
00007
00008 template<class T> class Vector {
00009 private:
00010     T* arr;
00011     int capacity{};
00012     int _size{};
00013
00014 public:
00015     Vector();
00016     explicit Vector(int capacity);
00017     Vector(const Vector<T>& other);
00018     ~Vector();
00019
00020     void push_back(T data);
00021     void pop_back();
00022     void insert(int index, T data);
00023     void erase(int index);
00024     void erase(T* position);
00025     void clear();
00026     void resize(int capacity);
00027     void assign(int capacity, T data);
00028     // void reserve();
00029     // void shrink_to_fit();
00030
00031     T& operator[](int index);
00032     Vector<T>& operator=(const Vector<T>& other);
00033
00034     [[nodiscard]] int getCapacity() const;
00035     [[nodiscard]] int size() const;
00036     [[nodiscard]] bool empty() const;
00037     T& at(int index) const;
00038     T& front() const;
00039     T& back() const;
00040     T* data() const;
00041     T* begin();
```

```

00042     T* end();
00043 };
00044
00045 template<class T>
00046 void Vector<T>::assign(int _capacity, T data) {
00047     this->clear();
00048     this->resize(_capacity);
00049     for (int i = 0; i < capacity; ++i) {
00050         this->arr[i] = data;
00051     }
00052 }
00053 }
00054
00055 template<class T>
00056 void Vector<T>::erase(T *position) {
00057     for (int i = 0; i < this->_size; ++i) {
00058         if (this->arr + i == position) {
00059             this->erase(i);
00060             break;
00061         }
00062     }
00063 }
00064
00065 template<class T>
00066 T *Vector<T>::end() {
00067     return this->arr + this->_size;
00068 }
00069
00070 template<class T>
00071 T *Vector<T>::begin() {
00072     return this->arr;
00073 }
00074
00075 template<class T>
00076 T *Vector<T>::data() const {
00077     return this->arr;
00078 }
00079
00080 template<class T>
00081 T &Vector<T>::back() const {
00082     return this->arr[this->_size - 1];
00083 }
00084
00085 template<class T>
00086 T &Vector<T>::front() const {
00087     return this->arr[0];
00088 }
00089
00090 template<class T>
00091 T &Vector<T>::at(int index) const {
00092     return this->arr[index];
00093 }
00094
00095 template<class T>
00096 bool Vector<T>::empty() const {
00097     return this->_size == 0;
00098 }
00099
00100 template<class T>
00101 int Vector<T>::size() const {
00102     return this->_size;
00103 }
00104
00105 template<class T>
00106 int Vector<T>::getCapacity() const {
00107     return this->capacity;
00108 }
00109
00110 template<class T>
00111 Vector<T> &Vector<T>::operator=(const Vector<T> &other) {
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 }
00123
00124 template<class T>
00125 T &Vector<T>::operator[](int index) {
00126     return this->arr[index];
00127 }
00128

```

```

00129 template<class T>
00130 void Vector<T>::resize(int _capacity) {
00131     this->_size = _capacity;
00132     if (_capacity > 0) {
00133         this->capacity = _capacity;
00134         T* temp = new T[this->capacity];
00135         for (int i = 0; i < this->_size; i++) {
00136             temp[i] = this->arr[i];
00137         }
00138         delete[] this->arr;
00139         this->arr = temp;
00140     }
00141 }
00142
00143 template<class T>
00144 void Vector<T>::clear() {
00145     this->_size = 0;
00146 }
00147
00148 template<class T>
00149 void Vector<T>::erase(int index) {
00150     if (index >= 0 && index < this->_size) {
00151         for (int i = index; i < this->_size - 1; i++) {
00152             this->arr[i] = this->arr[i + 1];
00153         }
00154         this->_size--;
00155     }
00156 }
00157
00158 template<class T>
00159 void Vector<T>::insert(int index, T data) {
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 }
00177
00178 template<class T>
00179 void Vector<T>::pop_back() {
00180     if (this->_size > 0) {
00181         this->_size--;
00182     }
00183 }
00184
00185 template<class T>
00186 void Vector<T>::push_back(T data) {
00187     if (this->_size >= this->capacity) {
00188         this->capacity *= 2;
00189         T* temp = new T[this->capacity];
00190         for (int i = 0; i < this->_size; i++) {
00191             temp[i] = this->arr[i];
00192         }
00193         delete[] this->arr;
00194         this->arr = temp;
00195     }
00196     this->arr[this->_size] = data;
00197     this->_size++;
00198 }
00199
00200 template<class T>
00201 Vector<T>::Vector(const Vector<T> &other) {
00202     this->capacity = other.capacity;
00203     this->_size = other._size;
00204     this->arr = new T[this->capacity];
00205     for (int i = 0; i < this->_size; i++) {
00206         this->arr[i] = other.arr[i];
00207     }
00208 }
00209
00210 template<class T>
00211 Vector<T>::~Vector() {
00212     delete[] this->arr;
00213 }
00214
00215 template<class T>

```

```

00216 Vector<T>::Vector(int capacity) {
00217     this->capacity = capacity;
00218     this->_size = capacity;
00219     this->arr = new T[this->capacity];
00220 }
00221
00222 template<class T>
00223 Vector<T>::Vector() {
00224     this->capacity = 10;
00225     this->_size = 0;
00226     this->arr = new T[this->capacity];
00227 }
00228
00229 #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.](#)

```

00001 //
00002 // Created by dirii on 08/04/2023.
00003 //
00004
00005 #include "Arrow.hpp"
00006
00007 Arrow::Arrow(sf::RenderWindow *window, sf::Vector2f start, sf::Vector2f end) : BaseDraw(window) {
00008     this->points[0] = start;
00009     this->points[1] = end;
00010
00011     this->arrowTexture[0].loadFromFile("../assets/arrow/arrow_black.png");
00012     this->arrowTexture[1].loadFromFile("../assets/arrow/arrow_orange.png");
00013
00014     this->arrowTexture[0].setSmooth(true);
00015     this->arrowTexture[1].setSmooth(true);
00016
00017     this->arrowSprite.setTexture(this->arrowTexture[0]);
00018     sf::Vector2i topLeftCorner(
00019         static_cast<int>(this->arrowTexture[0].getSize().x / 2.0 - constants::Arrow::sizeArrow.x /
00020             2.0),
00021         static_cast<int>(this->arrowTexture[0].getSize().y / 2.0 - constants::Arrow::sizeArrow.y /
00022             2.0)
00023     );
00024     this->arrowSprite.setTextureRect(sf::IntRect(
00025         topLeftCorner.x,
00026         topLeftCorner.y,
00027         constants::Arrow::sizeArrow.x,
00028         constants::Arrow::sizeArrow.y
00029     ));
00030     this->autoScale();
00031     this->autoRotate();
00032
00033     // this->rectangleTexture[0].loadFromFile("../assets/rectangle/rectangle_black.png");
00034     // this->rectangleTexture[1].loadFromFile("../assets/rectangle/rectangle_orange.png");
00035     // topLeftCorner = sf::Vector2i(
00036     //     static_cast<int>(this->rectangleTexture[0].getSize().x / 2.0 -
00037     //         constants::Arrow::sizeRectangle.x / 2.0),
00038     //     static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
00039     //         constants::Arrow::sizeRectangle.y / 2.0)
00040     // );
00041     // this->rectangleSprite.setTexture(this->rectangleTexture[0]);
00042     // this->rectangleSprite.setTextureRect(sf::IntRect(
00043     //     topLeftCorner.x,
00044     //     topLeftCorner.y,
00045     //     constants::Arrow::sizeRectangle.x,
00046     //     constants::Arrow::sizeRectangle.y
00047     // ));
00048     // this->rectangleSprite.setScale(
00049     //     constants::Arrow::defaultScaleRectangle.x,
00050     //     constants::Arrow::defaultScaleRectangle.y
00051     // );

```

```

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 }
00058
00059 void Arrow::render() {
00060     this->window->draw(this->arrowSprite);
00061     //      this->window->draw(this->rectangleSprite);
00062 }
00063
00064 void Arrow::toggleActiveColor() {
00065     this->arrowSprite.setTexture(this->arrowTexture[1]);
00066     //      this->rectangleSprite.setTexture(this->rectangleTexture[1]);
00067 }
00068
00069 void Arrow::resetColor() {
00070     this->arrowSprite.setTexture(this->arrowTexture[0]);
00071     //      this->rectangleSprite.setTexture(this->rectangleTexture[0]);
00072 }
00073
00074 void Arrow::setPositions(sf::Vector2f start, sf::Vector2f end, bool needSetMid) {
00075     this->points[0] = start;
00076     this->points[1] = end;
00077     if (needSetMid) {
00078         this->hasSetMid = false;
00079         this->setMid();
00080     }
00081     else {
00082         this->arrowSprite.setPosition(this->points[0]);
00083         this->autoScale();
00084         this->autoRotate();
00085     }
00086 }
00087
00088 void Arrow::autoRotate() {
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 }
00093
00094 void Arrow::autoScale() {
00095     this->length = static_cast<float>(
00096         sqrt(
00097             pow(this->points[1].x - this->points[0].x, 2) + pow(this->points[1].y -
00098                 this->points[0].y, 2)
00099             ) - constants::NodeInfo::radius - 2.f
00100         );
00101     this->arrowSprite.setScale(
00102         this->length / this->arrowSprite.getLocalBounds().width,
00103         constants::Arrow::defaultScaleArrow.y
00104     );
00105     this->arrowSprite.setOrigin(
00106         0,
00107         this->arrowSprite.getLocalBounds().height / 2.0f
00108     );
00109     this->arrowSprite.setPosition(this->points[0]);
00110 }
00111 void Arrow::setMid() {
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 }
00120
00121 void Arrow::setStart(sf::Vector2f start, bool needSetMid) {
00122     this->setPositions(start, this->points[1], needSetMid);
00123 }
00124
00125 void Arrow::hide() {
00126     sf::Color tmp = this->arrowSprite.getColor();
00127     tmp.a = 0;
00128     this->arrowSprite.setColor(tmp);
00129 }
00130
00131 void Arrow::show() {
00132     sf::Color tmp = this->arrowSprite.getColor();
00133     tmp.a = 255;
00134     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.](#)

```
00001 //
00002 // Created by dirii on 08/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_ARROW_HPP
00006 #define VISUALGO_CS162_ARROW_HPP
00007
00008 #include <cmath>
00009 #include <SFML/Graphics.hpp>
00010 #include "BaseDraw.hpp"
00011 #include "Constants.hpp"
00012
00013 class Arrow : public BaseDraw{
00014 protected:
00015     sf::Vector2f points[2];
00016     sf::Texture arrowTexture[2];
00017     sf::Sprite arrowSprite;
00018     float length;
00019     bool hasSetMid;
00020
00021 public:
00022     Arrow(sf::RenderWindow* window, sf::Vector2f start, sf::Vector2f end);
00023     void render() override;
00024     void toggleActiveColor();
00025     void resetColor();
00026     void setStart(sf::Vector2f start, bool needSetMid);
00027     void setPositions(sf::Vector2f start, sf::Vector2f end, bool needSetMid);
00028     void setMid();
00029     void autoRotate();
00030     void autoScale();
00031
00032     void hide();
00033     void show();
00034 };
00035
00036 #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.](#)

```

00001 //
00002 // Created by dirii on 26/04/2023.
00003 //
00004
00005 #include "BackArrow.hpp"
00006
00007 BackArrow::BackArrow(sf::RenderWindow *window, sf::Vector2f start, sf::Vector2f end) :
    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 -
00030             constants::Arrow::sizeRectangle.x / 2.0),
00031         static_cast<int>(this->rectangleTexture[0].getSize().y / 2.0 -
00032             constants::Arrow::sizeRectangle.y / 2.0)
00033     );
00034     for (auto & rectangleSprite : this->rectangleSprites) {
00035         rectangleSprite.setTexture(this->rectangleTexture[0]);
00036         rectangleSprite.setTextureRect(sf::IntRect(
00037             topLeftCorner.x,
00038             topLeftCorner.y,
00039             constants::Arrow::sizeRectangle.x,
00040             constants::Arrow::sizeRectangle.y
00041         ));
00042     }
00043     this->setPosition(start, end);
00044
00045 void BackArrow::render() {
00046     if (this->isShow) {
00047         this->window->draw(this->rectangleSprites[0]);
00048         this->window->draw(this->rectangleSprites[1]);
00049         this->arrow->render();
00050     }
00051 }
00052
00053 void BackArrow::show() {
00054     this->isShow = true;
00055 }
00056
00057 void BackArrow::hide() {
00058     this->isShow = false;
00059 }
00060
00061 void BackArrow::toggleActiveColorNode() {
00062     this->rectangleSprites[0].setTexture(this->rectangleTexture[1]);
00063     this->rectangleSprites[1].setTexture(this->rectangleTexture[0]);
00064     this->arrow->toggleActiveColor();
00065 }
00066
00067 void BackArrow::resetColor() {
00068     this->rectangleSprites[0].setTexture(this->rectangleTexture[0]);
00069     this->rectangleSprites[1].setTexture(this->rectangleTexture[1]);
00070     this->arrow->resetColor();
00071 }
00072
00073 void BackArrow::setPosition(sf::Vector2f start, sf::Vector2f end) {
00074     this->points[0] = end;
00075     this->points[1] = start;
00076     if (end == start) {
00077         this->hide();
00078         return;
00079     }
}

```

```

00080     this->points[2] = sf::Vector2f(
00081         this->points[0].x,
00082         this->points[0].y - constants::NodeInfo::offsetX
00083     );
00084     this->points[3] = sf::Vector2f(
00085         this->points[1].x,
00086         this->points[2].y
00087     );
00088     this->arrow->setPositions(this->points[2], this->points[0], false);
00089     this->autoRotate();
00090     this->autoScale();
00091 }
00092
00093 void BackArrow::autoScale() {
00094     float length = sqrtf(
00095         powf(this->points[3].x - this->points[2].x, 2) + powf(this->points[3].y -
00096             this->points[2].y, 2)
00097     );
00098     this->rectangleSprites[0].setScale(
00099         length / this->rectangleSprites[0].getLocalBounds().width,
00100         constants::Arrow::defaultScaleRectangle.y
00101     );
00102     length = sqrtf(
00103         powf(this->points[3].x - this->points[1].x, 2) + powf(this->points[3].y -
00104             this->points[1].y, 2)
00105     );
00106     this->rectangleSprites[1].setScale(
00107         length / this->rectangleSprites[1].getLocalBounds().width,
00108         constants::Arrow::defaultScaleRectangle.y
00109     );
00110     this->rectangleSprites[0].setOrigin(
00111         this->rectangleSprites[0].getLocalBounds().width / 2.0f,
00112         0
00113     );
00114     this->rectangleSprites[1].setOrigin(
00115         this->rectangleSprites[1].getLocalBounds().width,
00116         this->rectangleSprites[1].getLocalBounds().height / 2.0f
00117     );
00118     this->rectangleSprites[0].setPosition(
00119         (this->points[3].x + this->points[2].x) / 2.0f,
00120         (this->points[3].y + this->points[2].y) / 2.0f
00121     );
00122     this->rectangleSprites[1].setPosition(this->points[1]);
00123 }
00124
00125 void BackArrow::autoRotate() {
00126     sf::Vector2f vector2point = this->points[3] - this->points[2];
00127     float angle = atan2f(vector2point.y, vector2point.x) * 180.0f / (float)M_PI;
00128     this->rectangleSprites[0].setRotation(angle);
00129     vector2point = this->points[1] - this->points[3];
00130     angle = atan2f(vector2point.y, vector2point.x) * 180.0f / (float)M_PI;
00131     this->rectangleSprites[1].setRotation(angle);
00132 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 26/04/2023.
00003 //

```

```

00004
00005 #ifndef VISUALGO_CS162_BACKARROW_HPP
00006 #define VISUALGO_CS162_BACKARROW_HPP
00007
00008 #include "Arrow.hpp"
00009 #include "Constants.hpp"
00010
00011 class BackArrow : public BaseDraw {
00012 private:
00013     sf::Vector2f points[4];
00014     sf::Texture rectangleTexture[2];
00015     sf::Sprite rectangleSprites[2];
00016     Arrow* arrow;
00017     bool isShow;
00018
00019 public:
00020     BackArrow(sf::RenderWindow* window, sf::Vector2f start, sf::Vector2f end);
00021     void render() override;
00022
00023     void autoScale();
00024     void autoRotate();
00025
00026     void toggleActiveColorNode();
00027     void resetColor();
00028
00029     void setPosition(sf::Vector2f start, sf::Vector2f end);
00030
00031     void show();
00032     void hide();
00033 };
00034
00035 #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.](#)

```

00001 //
00002 // Created by dirii on 08/04/2023.
00003 //
00004
00005 #include "BaseDraw.hpp"
00006
00007 BaseDraw::BaseDraw(sf::RenderWindow *window) {
00008     this->window = window;
00009 }

```

## 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.](#)

```
00001 //
00002 // Created by dirii on 08/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_BASEDRAW_HPP
00006 #define VISUALGO_CS162_BASEDRAW_HPP
00007
00008 #include <SFML/Graphics.hpp>
00009
00010 class BaseDraw{
00011 protected:
00012     sf::RenderWindow* window;
00013
00014 public:
00015     explicit BaseDraw(sf::RenderWindow* window);
00016
00017
00018     virtual void render() = 0;
00019 };
00020
00021 #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.](#)

```
00001 //
00002 // Created by dirii on 08/04/2023.
00003 //
00004
00005 #include "NodeInfo.hpp"
00006
00007 NodeInfo::NodeInfo(sf::RenderWindow *window, std::string value, sf::Vector2f position, bool _isDLL) :
    BaseDraw(window) {
00008     this->values[0] = value;
00009     this->values[1] = value;
00010
00011     this->positions[(int)TypeNode::Normal] = position;
00012     this->positions[(int)TypeNode::Effective] = position;
00013     this->positions[(int)TypeNode::Outside] = sf::Vector2f(
00014         position.x,
00015         position.y + constants::NodeInfo::offsetY
00016     );
00017
00018     this->isDLL = _isDLL;
00019
00020     this->statusNode = StatusNode::InChain;
00021
00022     this->node = new SingleNode(window, std::move(value), this->positions[(int)TypeNode::Normal]);
00023
00024     for (auto &arrow : this->arrows)
00025         arrow[(int)ArrowType::LEFT] = arrow[(int)ArrowType::RIGHT] = nullptr;
00026
00027     this->isPrintPreVal = this->isPrintNormal = false;
00028
00029     this->title.setFont(this->node->font);
00030     this->title.setCharacterSize(constants::TitleNode::fontSize);
00031     this->title.setFillColor(constants::titleGreen);
00032     this->title.setString("");
00033 }
00034
00035 void NodeInfo::render() {
00036     if (this->statusNode == StatusNode::Visible)
00037         return;
00038 }
```

```

00039     if (this->isDLL && this->statusNode == StatusNode::InChain){
00040         if (this->arrows[1][(int)ArrowType::LEFT])
00041             this->arrows[1][(int)ArrowType::LEFT]->render();
00042         if (this->arrows[1][(int)ArrowType::RIGHT])
00043             this->arrows[1][(int)ArrowType::RIGHT]->render();
00044     } else {
00045         if (this->arrows[0][(int)ArrowType::LEFT])
00046             this->arrows[0][(int)ArrowType::LEFT]->render();
00047         if (this->arrows[0][(int)ArrowType::RIGHT])
00048             this->arrows[0][(int)ArrowType::RIGHT]->render();
00049     }
00050     this->node->render();
00051     this->window->draw(this->title);
00052 }
00053
00054 void NodeInfo::initArrow(NodeInfo::ArrowType type, sf::Vector2f start, sf::Vector2f end) {
00055     this->arrows[1][(int)type] = new Arrow(this->window, start, end);
00056     this->arrows[1][(int)type]->setMid();
00057     this->arrows[0][(int)type] = new Arrow(this->window, start, end);
00058 }
00059
00060 void NodeInfo::toggleActiveColorNode() {
00061     this->node->toggleActiveColor();
00062 }
00063
00064 void NodeInfo::toggleActiveColorArrow(NodeInfo::ArrowType type) {
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 }
00070
00071 void NodeInfo::resetColorNode() {
00072     this->node->resetColor();
00073 }
00074
00075 void NodeInfo::resetColorArrow(NodeInfo::ArrowType type) {
00076     if (this->arrows[0][(int)type])
00077         this->arrows[0][(int)type]->resetColor();
00078     if (this->arrows[1][(int)type])
00079         this->arrows[1][(int)type]->resetColor();
00080 }
00081
00082 void NodeInfo::reset() {
00083     this->resetColorNode();
00084     this->resetColorArrow(ArrowType::LEFT);
00085     this->resetColorArrow(ArrowType::RIGHT);
00086     this->resetTitle();
00087     this->isPrintNormal = this->isPrintPreVal = false;
00088     this->statusNode = StatusNode::InChain;
00089     this->show(ArrowType::LEFT);
00090     this->show(ArrowType::RIGHT);
00091 }
00092
00093 // require update() before calling this function
00094 sf::Vector2f NodeInfo::getPosition() {
00095     this->updateNode(); // ?
00096     return this->node->getPosition();
00097 }
00098
00099 void NodeInfo::reInitPos(int index) {
00100     this->positions[(int)TypeNode::Normal] = sf::Vector2f(
00101         constants::NodeInfo::originNode.x + static_cast<float>(index) *
00102         constants::NodeInfo::offsetX,
00103         constants::NodeInfo::originNode.y
00104     );
00105     this->positions[(int)TypeNode::Outside] = sf::Vector2f(
00106         this->positions[(int)TypeNode::Effective].x,
00107         this->positions[(int)TypeNode::Effective].y + constants::NodeInfo::offsetY
00108     );
00109 }
00110
00111 void NodeInfo::setPrintPreVal() {
00112     this->isPrintPreVal = true;
00113 }
00114
00115 void NodeInfo::setPrintNormal() {
00116     this->isPrintNormal = true;
00117 }
00118
00119 void NodeInfo::setNodeOutside() {
00120     this->statusNode = StatusNode::OutChain;
00121 }
00122
00123 void NodeInfo::setNodeInChain() {
00124     this->statusNode = StatusNode::InChain;
00125 }

```

```

00125
00126 void NodeInfo::setNodeVisible() {
00127     this->statusNode = StatusNode::Visible;
00128 }
00129
00130 // require calculate effective positions of a chain before calling this function
00131 void NodeInfo::updateNode() {
00132     if (this->statusNode == StatusNode::Visible)
00133         return;
00134
00135     if (this->statusNode == StatusNode::InChain) {
00136         if (this->isPrintNormal) {
00137             this->node->setPosition(this->positions[(int)TypeNode::Normal]);
00138         } else {
00139             this->node->setPosition(this->positions[(int)TypeNode::Effective]);
00140         }
00141     } else {
00142         this->node->setPosition(this->positions[(int)TypeNode::Outside]);
00143     }
00144
00145     if (this->isPrintPreVal) {
00146         this->node->setText(this->values[1]);
00147     } else {
00148         this->node->setText(this->values[0]);
00149     }
00150 }
00151
00152 void NodeInfo::updateArrows(ArrowType type, sf::Vector2f end){
00153     if (this->arrows[0][(int)type])
00154         this->arrows[0][(int)type]->setPositions(this->node->getPosition(), end, false);
00155
00156     if (this->arrows[1][(int)type])
00157         this->arrows[1][(int)type]->setPositions(this->node->getPosition(), end, true);
00158 }
00159
00160 void NodeInfo::reInitPreVal() {
00161     this->values[1] = this->values[0];
00162 }
00163
00164 NodeInfo::StatusNode NodeInfo::getStatusNode() {
00165     return this->statusNode;
00166 }
00167
00168 void NodeInfo::setEffectivePosition(sf::Vector2f start) {
00169     this->positions[(int)TypeNode::Effective] = start;
00170 }
00171
00172 void NodeInfo::setArrows(NodeInfo::ArrowType type, sf::Vector2f start, sf::Vector2f end) {
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 }
00178
00179 void NodeInfo::hide(NodeInfo::ArrowType type) {
00180     if (this->arrows[0][(int)type])
00181         this->arrows[0][(int)type]->hide();
00182     if (this->arrows[1][(int)type])
00183         this->arrows[1][(int)type]->hide();
00184 }
00185
00186 void NodeInfo::show(NodeInfo::ArrowType type) {
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 }
00192
00193 NodeInfo::~NodeInfo() {
00194     delete this->node;
00195     for (auto & arrow : this->arrows) {
00196         for (auto & j : arrow) {
00197             delete j;
00198         }
00199     }
00200 }
00201
00202 void NodeInfo::setValue(std::string value) {
00203     this->values[0] = std::move(value);
00204 }
00205
00206 std::string NodeInfo::getValue() {
00207     return this->values[0];
00208 }
00209
00210 void NodeInfo::setTitle(const std::string& _title) {
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 }
00226
00227 void NodeInfo::resetTitle() {
00228     this->title.setString("");
00229 }
00230
00231 void NodeInfo::destroyArrow(NodeInfo::ArrowType type) {
00232     if (this->arrows[0][(int)type])
00233         delete this->arrows[0][(int)type];
00234     if (this->arrows[1][(int)type])
00235         delete this->arrows[1][(int)type];
00236     this->arrows[0][(int)type] = nullptr;
00237     this->arrows[1][(int)type] = nullptr;
00238 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 08/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_NODEINFO_HPP
00006 #define VISUALGO_CS162_NODEINFO_HPP
00007
00008 #include "BaseDraw.hpp"
00009 #include "SingleNode.hpp"
00010 #include "Arrow.hpp"
00011
00012 class NodeInfo : public BaseDraw {
00013 public:
00014     enum class ArrowType {
00015         LEFT,
00016         RIGHT
00017     };
00018
00019     enum class StatusNode{
00020         InChain,
00021         OutChain,
00022         Visible
00023     };
00024
00025     enum class TypeNode{
00026         Normal,
00027         Outside,
00028         Effective

```

```

00029     };
00030
00031     NodeInfo(sf::RenderWindow* window, std::string value, sf::Vector2f position, bool _isDLL);
00032     ~NodeInfo();
00033     void updateNode();
00034     void updateArrows(ArrowType type, sf::Vector2f end);
00035     void render() override;
00036
00037     void initArrow(ArrowType type, sf::Vector2f start, sf::Vector2f end);
00038     void destroyArrow(ArrowType type);
00039
00040     void reInitPos(int index);
00041     void reInitPreVal();
00042
00043     void setEffectivePosition(sf::Vector2f start);
00044     void setArrows(ArrowType type, sf::Vector2f start, sf::Vector2f end);
00045     void setValue(std::string value);
00046
00047     sf::Vector2f getPosition();
00048     std::string getValue();
00049
00050     void toggleActiveColorNode();
00051     void toggleActiveColorArrow(ArrowType type);
00052
00053     void setPrintPreVal();
00054     void setPrintNormal();
00055
00056     void setNodeInChain();
00057     void setNodeOutside();
00058     void setNodeVisible();
00059
00060     void setTitle(const std::string& title);
00061
00062     void hide(ArrowType type);
00063     void show(ArrowType type);
00064
00065     StatusNode getStatusNode();
00066
00067     void resetColorNode();
00068     void resetColorArrow(ArrowType type);
00069     void resetTitle();
00070
00071     void reset();
00072
00073 private:
00074     sf::Vector2f positions[3];
00075     SingleNode* node;
00076     Arrow* arrows[2][2];
00077     std::string values[2];
00078     sf::Text title;
00079
00080     StatusNode statusNode;
00081     bool isPrintPreVal, isDLL, isPrintNormal;
00082 };
00083
00084 #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.](#)

```

00001 //
00002 // Created by dirii on 10/04/2023.
00003 //
00004
00005 #include "SingleNode.hpp"
00006
00007 SingleNode::SingleNode(sf::RenderWindow *window, std::string value, sf::Vector2f position) :
    BaseDraw(window) {
00008     this->value = std::move(value);
00009

```



```

00010     this->circle.setRadius(constants::NodeInfo::radius);
00011     this->circle.setFill(sf::Color::White);
00012     this->circle.setOutlineThickness(constants::NodeInfo::outlineThickness);
00013     this->circle.setOutlineColor(sf::Color::Black);
00014     this->circle.setPointCount(constants::NodeInfo::pointCount);
00015     sf::FloatRect bounds = this->circle.getLocalBounds();
00016     this->circle.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00017     this->circle.setPosition(position);
00018
00019     this->font.loadFromFile(constants::fontPath);
00020     this->label.setFont(this->font);
00021     this->label.setString(this->value);
00022     this->label.setCharacterSize(constants::NodeInfo::fontSize);
00023     this->label.setFill(sf::Color::Black);
00024     bounds = this->label.getLocalBounds();
00025     this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00026     this->label.setPosition(position);
00027 }
00028
00029 void SingleNode::render() {
00030     this->window->draw(this->circle);
00031     this->window->draw(this->label);
00032 }
00033
00034 void SingleNode::toggleActiveColor() {
00035     this->circle.setOutlineColor(constants::normalGreen);
00036 }
00037
00038 void SingleNode::resetColor() {
00039     this->circle.setOutlineColor(sf::Color::Black);
00040 }
00041
00042 void SingleNode::setPosition(sf::Vector2f position) {
00043     this->circle.setPosition(position);
00044     this->label.setPosition(position);
00045 }
00046
00047 sf::Vector2f SingleNode::getPosition() {
00048     return this->circle.getPosition();
00049 }
00050
00051 void SingleNode::setText(std::string _value) {
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 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 10/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_SINGLENODE_HPP
00006 #define VISUALGO_CS162_SINGLENODE_HPP
00007

```

```

00008 #include "Constants.hpp"
00009 #include "BaseDraw.hpp"
00010 #include <iostream>
00011
00012 class SingleNode : public BaseDraw{
00013 private:
00014     sf::CircleShape circle;
00015     sf::Text label;
00016     std::string value;
00017
00018 public:
00019     sf::Font font;
00020
00021     SingleNode(sf::RenderWindow* window, std::string value, sf::Vector2f position);
00022     void render() override;
00023     void toggleActiveColor();
00024     void resetColor();
00025     void setText(std::string _value);
00026     void setPosition(sf::Vector2f position);
00027     sf::Vector2f getPosition();
00028 };
00029
00030 #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.](#)

```

00001 //
00002 // Created by dirii on 28/04/2023.
00003 //
00004
00005 #include "Square.hpp"
00006
00007 Square::Square(sf::RenderWindow *window, std::string value, sf::Vector2f position)
00008     : BaseDraw(window) {
00009     this->value = std::move(value);
00010
00011     this->square.setSize(sf::Vector2f(constants::Square::length, constants::Square::length));
00012     this->square.setFillColor(sf::Color::White);
00013     this->square.setOutlineThickness(constants::Square::outlineThickness);
00014     this->square.setOutlineColor(sf::Color::Black);
00015     sf::FloatRect bounds = this->square.getLocalBounds();
00016     this->square.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00017     this->square.setPosition(position);
00018
00019     this->font.loadFromFile(constants::fontPath);
00020     this->label.setFont(this->font);
00021     this->label.setString(this->value);
00022     this->label.setCharacterSize(constants::Square::fontSize);
00023     this->label.setFillColor(sf::Color::Black);
00024     bounds = this->label.getLocalBounds();
00025     this->label.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00026     this->label.setPosition(position);
00027 }
00028
00029 void Square::render() {
00030     switch (this->status) {
00031         case Status::active:
00032             this->square.setOutlineColor(constants::normalGreen);
00033             break;
00034         case Status::inactive:
00035             this->square.setOutlineColor(sf::Color::Black);
00036             break;
00037         case Status::chosen:
00038             this->square.setOutlineColor(constants::clickGreen);
00039             break;
00040     }
00041     this->window->draw(this->square);
00042     this->window->draw(this->label);
00043 }

```

```

00044
00045 void Square::resetColor() {
00046     this->status = Status::inactive;
00047 }
00048
00049 void Square::setText(std::string _value) {
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 }
00056
00057 void Square::setPosition(sf::Vector2f position) {
00058     this->square.setPosition(position);
00059     this->label.setPosition(position);
00060 }
00061
00062 sf::Vector2f Square::getPosition() {
00063     return this->square.getPosition();
00064 }
00065
00066 void Square::setStatus(Square::Status _status) {
00067     this->status = _status;
00068 }
00069
00070 Square::Status Square::getStatus() {
00071     return this->status;
00072 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 28/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_SQUARE_HPP
00006 #define VISUALGO_CS162_SQUARE_HPP
00007
00008 #include "Constants.hpp"
00009 #include "BaseDraw.hpp"
00010
00011 class Square : BaseDraw{
00012 public:
00013     enum class Status{
00014         inactive,
00015         active,
00016         chosen,
00017         hidden
00018     };
00019
00020     sf::Font font;
00021
00022     Square(sf::RenderWindow *window, std::string value, sf::Vector2f position);
00023     void render() override;
00024
00025     void setStatus(Status _status);
00026     void resetColor();
00027     Status getStatus();

```

```

00028
00029     void setText(std::string _value);
00030     void setPosition(sf::Vector2f position);
00031     sf::Vector2f getPosition();
00032
00033 private:
00034     sf::RectangleShape square;
00035     sf::Text label;
00036     std::string value;
00037     Status status = Status::inactive;
00038 };
00039
00040 #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.](#)

```

00001 //
00002 // Created by dirii on 28/04/2023.
00003 //
00004
00005 #include "SquareInfo.hpp"
00006
00007 SquareInfo::SquareInfo(sf::RenderWindow *window, std::string value, sf::Vector2f position) :
    BaseDraw(window) {
00008     this->position = position;
00009     this->square = new Square(window, value, position);
00010     this->values[0] = std::move(value);
00011     this->values[1] = "";
00012     this->isPrintPreVal = false;
00013
00014     this->title.setFont(this->square->font);
00015     this->title.setCharacterSize(20);
00016     this->title.setFillColor(sf::Color::Black);
00017 }
00018
00019 void SquareInfo::render() {
00020     if (this->square->getStatus() != Square::Status::hidden) {
00021         this->square->render();
00022         this->window->draw(this->title);
00023     }
00024 }
00025
00026 void SquareInfo::setValue(std::string value) {
00027     this->values[1] = this->values[0];
00028     this->values[0] = std::move(value);
00029 }
00030
00031 void SquareInfo::update() {
00032     if (this->isPrintPreVal)
00033         this->square->setText(this->values[1]);
00034     else
00035         this->square->setText(this->values[0]);
00036 }
00037
00038 void SquareInfo::setTitle(const std::string& _title) {
00039     this->title.setString(_title);
00040     sf::FloatRect bounds = this->title.getLocalBounds();
00041     this->title.setOrigin(bounds.left + bounds.width / 2.0f, bounds.top + bounds.height / 2.0f);
00042     this->title.setPosition(this->position.x, this->position.y + constants::TitleNode::offsetY);
00043 }
00044
00045 void SquareInfo::resetTitle() {
00046     this->title.setString("");
00047 }
00048
00049 void SquareInfo::reset() {
00050     this->resetTitle();
00051     this->square->resetColor();
00052     this->isPrintPreVal = false;

```

```

00053 }
00054
00055 void SquareInfo::setStatus(Square::Status _status) {
00056     this->square->setStatus(_status);
00057 }
00058
00059 std::string SquareInfo::getValue() {
00060     return this->values[0];
00061 }
00062
00063 void SquareInfo::setPrintPreVal(bool _isPrintPreVal) {
00064     this->isPrintPreVal = _isPrintPreVal;
00065 }
00066
00067 Square::Status SquareInfo::getStatus() {
00068     return this->square->getStatus();
00069 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 28/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_SQUAREINFO_HPP
00006 #define VISUALGO_CS162_SQUAREINFO_HPP
00007
00008 #include "Square.hpp"
00009
00010 class SquareInfo : public BaseDraw {
00011 public:
00012     SquareInfo(sf::RenderWindow *window, std::string value, sf::Vector2f position);
00013     ~SquareInfo() = default;
00014     void update();
00015     void render() override;
00016
00017     void setValue(std::string value);
00018     void setTitle(const std::string& _title);
00019     void setStatus(Square::Status _status);
00020     void setPrintPreVal(bool _isPrintPreVal);
00021
00022     std::string getValue();
00023     Square::Status getStatus();
00024
00025     void resetTitle();
00026     void reset();
00027
00028 private:
00029     sf::Vector2f position;
00030     Square* square;
00031     std::string values[2];
00032     sf::Text title;
00033
00034     bool isPrintPreVal;
00035 };
00036
00037 #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.](#)

```
00001 //
00002 // Created by dirii on 29/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_ALLSCENES_HPP
00006 #define VISUALGO_CS162_ALLSCENES_HPP
00007
00008 #include "MainMenu.hpp"
00009 #include "SLLScene.hpp"
00010 #include "DLLScene.hpp"
00011 #include "CLLScene.hpp"
00012 #include "StackScene.hpp"
00013 #include "QueueScene.hpp"
00014 #include "StaticArrayScene.hpp"
00015 #include "DynamicArrayScene.hpp"
00016
00017 #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.](#)

```
00001 //
00002 // Created by dirii on 25/03/2023.
00003 //
00004
00005 #include "BaseScene.hpp"
00006
00007 void BaseScene::setWindow(sf::RenderWindow *window) {
00008     this->window = window;
00009 }
00010
00011 void BaseScene::createModeButton(sf::Vector2f position, std::string textString) {
00012     this->modeButton = new Button(
00013         this->window,
00014         position,
00015         constants::modeButtonSize,
00016         textString,
00017         textString,
00018         constants::sizeTextModeButton,
00019         sf::Color::Black,
00020         constants::normalGray,
00021         constants::hoverGray,
```

```

00022         constants::clickGray
00023     );
00024 }
00025
00026 BaseScene::BaseScene(sf::RenderWindow *window) {
00027     this->setWindow(window);
00028     this->isMenuOpen = false;
00029     this->isDemoCodeOpen = false;
00030
00031     this->controlMenu = new ControlMenu(this->window);
00032 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 23/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_BASESCENE_HPP
00006 #define VISUALGO_CS162_BASESCENE_HPP
00007
00008 #include <SFML/Graphics.hpp>
00009 #include "stuff/button.hpp"
00010 #include "ControlMenu.hpp"
00011
00012 class BaseScene{
00013 protected:
00014     sf::RenderWindow* window{};
00015     ControlMenu* controlMenu;
00016
00017     void setWindow(sf::RenderWindow* window);
00018 public:
00019     Button* modeButton{};
00020     bool isMenuOpen{}, isDemoCodeOpen{};
00021
00022     explicit BaseScene(sf::RenderWindow* window);
00023
00024     void createModeButton(sf::Vector2f position, std::string textString);
00025
00026     virtual void pollEvent(sf::Event event, sf::Vector2f mousePosView) = 0;
00027     virtual void update() = 0;
00028     virtual void render() = 0;
00029 };
00030
00031 #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.](#)

```

00001 //
00002 // Created by dirii on 28/03/2023.
00003 //
00004
00005 #include "CLLScene.hpp"
00006
00007 CLLScene::CLLScene(sf::RenderWindow *window) : BaseScene(window) {
00008     this->init();
00009 }
00010
00011 void CLLScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00012     if (this->isMenuOpen)
00013         this->menu->pollEvents(event, mousePosView);
00014
00015     this->controlMenu->pollEvents(event, mousePosView);
00016 }
00017
00018 void CLLScene::update() {
00019     if (this->isMenuOpen) {
00020         this->menu->update();
00021
00022         constants::MenuLinkedList::Button status = this->menu->getActiveOptionsMenu();
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 ==
00035 constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
00036                     if (this->menu->createModeValue[1] == "None")
00037                         break;
00038                     std::vector<std::string> values;
00039                     std::string value = this->menu->createModeValue[1];
00040                     std::stringstream ss(value);
00041                     std::string token;
00042                     while (std::getline(ss, token, ',')) {
00043                         values.push_back(token);
00044                     }
00045                     this->linkedList->createLinkedList(values);
00046                 } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
00047                     if (this->menu->createModeValue[2] == "None")
00048                         break;
00049                     std::vector<std::string> values;
00050                     std::string value = this->menu->createModeValue[2];
00051                     std::stringstream ss(value);
00052                     std::string token;
00053                     while (std::getline(ss, token, ','))
00054                         values.push_back(token);
00055                     this->linkedList->createLinkedList(values);
00056                     this->menu->createModeValue[2] = "None";
00057                 }
00058                 this->controlMenu->reset();
00059                 break;
00060             case constants::MenuLinkedList::Button::ADD_BUTTON:
00061                 if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
00062 this->menu->addModeValue[0].empty())
00063                     break;
00064                 this->linkedList->addNode(
00065                     std::stoi(this->menu->addModeValue[0]),
00066                     this->menu->addModeValue[1],
00067                     this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00068                 );
00069                 std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
00070 << std::endl;
00071                 this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00072                 this->controlMenu->reset();
00073                 break;
00074             case constants::MenuLinkedList::Button::DELETE_BUTTON:
00075                 if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
00076                     break;
00077                 this->linkedList->deleteNode(
00078                     std::stoi(this->menu->deleteModeValue),
00079                     this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))

```



```

00080         );
00081
00082         std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00083         this->menu->deleteModeValue = "None";
00084         this->controlMenu->reset();
00085         break;
00086     case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00087         if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
00088             break;
00089
00090         this->linkedList->updateNode(
00091             std::stoi(this->menu->updateModeValue[0]),
00092             this->menu->updateModeValue[1],
00093             this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00094         );
00095
00096         std::cout << "Update: " << this->menu->updateModeValue[0] << " " <<
this->menu->updateModeValue[1] << std::endl;
00097         this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00098         this->controlMenu->reset();
00099         break;
00100     case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00101         if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00102             break;
00103
00104         this->linkedList->searchNode(
00105             this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
00106         );
00107
00108         std::cout << "Search: " << this->menu->searchModeValue << std::endl;
00109         this->menu->searchModeValue = "None";
00110         this->controlMenu->reset();
00111         break;
00112     }
00113 }
00114
00115 this->controlMenu->update();
00116
00117 this->linkedList->processControlMenu(this->controlMenu->getStatus());
00118 this->linkedList->setSpeed(this->controlMenu->getSpeed());
00119
00120 this->linkedList->update();
00121 }
00122
00123 void CLLScene::render() {
00124     if (this->isMenuOpen)
00125         this->menu->render();
00126
00127     if (this->isDemoCodeOpen)
00128         this->linkedList->renderHighlighter();
00129
00130     this->controlMenu->render();
00131     this->linkedList->render();
00132 }
00133
00134 void CLLScene::init() {
00135     this->menu = new MenuLinkedList(this->window);
00136     this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::CIRCULAR);
00137 }
00138
00139 void CLLScene::reset() {
00140     this->menu->resetActiveOptionsMenu();
00141 }
00142
00143 std::vector<EventAnimation> CLLScene::addModeEvents(int chosenNode) {
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 }
00287
00288 std::vector<EventAnimation> CLLScene::deleteModeEvents(int chosenNode) {
00289     this->linkedList->resetEvents();
00290     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00291         return {};
00292
00293     this->linkedList->initHighlighter(
00294         constants::Highlighter::SLL::CODES_PATH[1].second,
00295         constants::Highlighter::SLL::CODES_PATH[1].first
00296     );
00297
00298     std::vector<EventAnimation> events;
00299     EventAnimation event;
00300
00301     if (!chosenNode) {
00302         event.titleNodes.emplace_back(chosenNode, "head|temp");
00303         event.colorNodes.push_back(chosenNode);
00304         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00305         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00306         event.lines = {0, 1};
00307
00308         events.emplace_back(event);
00309
00310         if (this->linkedList->getSize() > 1) {
00311             event.reset();
00312             event.titleNodes = {
00313                 {chosenNode, "temp"},
00314                 {1, "head"}
00315             };
00316             event.colorNodes.push_back(1);
00317             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00318             event.isPrintNormal = true;
00319             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00320             event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
00321             event.lines = {2};
00322
00323             events.emplace_back(event);
00324         }
00325
00326         event.reset();
00327         event.titleNodes.emplace_back(1, "head");
00328         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00329         event.indexBackArrow = {this->linkedList->getSize() - 1, 1};
00330         event.lines = {3};
00331
00332         events.emplace_back(event);
00333     } else {
00334         event.reset();
00335         event.titleNodes.emplace_back(0, "head|current");
00336         event.colorNodes.push_back(0);
00337         event.statusChosenNode = NodeInfo::StatusNode::InChain;

```

```

00338     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00339     event.lines = {5};
00340
00341     events.emplace_back(event);
00342
00343     for (int i = 0; i < chosenNode; ++i) {
00344         event.reset();
00345         event.titleNodes = {
00346             {0, "head"},
00347             {i, "current"}
00348         };
00349         event.colorNodes.push_back(i);
00350         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00351         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00352         event.lines = {6};
00353
00354         events.emplace_back(event);
00355
00356         if (i == chosenNode - 1) break;
00357
00358         event.reset();
00359         event.titleNodes = {
00360             {0, "head"},
00361             {i, "current"}
00362         };
00363         event.colorNodes.push_back(i);
00364         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00365         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00366         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00367         event.lines = {7};
00368
00369         events.emplace_back(event);
00370     }
00371
00372     event.reset();
00373     event.titleNodes = {
00374         {0, "head"},
00375         {chosenNode, "temp"},
00376         {chosenNode - 1, "current"}
00377     };
00378     event.colorNodes.push_back(chosenNode);
00379     event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00380     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00381     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00382     event.lines = {8};
00383
00384     events.emplace_back(event);
00385
00386     if (chosenNode != this->linkedList->getSize() - 1) {
00387         event.reset();
00388         event.titleNodes = {
00389             {0, "head"},
00390             {chosenNode, "temp"},
00391             {chosenNode - 1, "current"}
00392         };
00393         event.colorNodes.push_back(chosenNode);
00394         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00395         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00396         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00397         event.isPrintNormal = true;
00398         event.lines = {9};
00399
00400         events.emplace_back(event);
00401
00402         event.reset();
00403         event.titleNodes.emplace_back(0, "head");
00404         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00405         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00406         event.lines = {10};
00407
00408         events.emplace_back(event);
00409     } else {
00410         event.reset();
00411         event.titleNodes = {
00412             {0, "head"},
00413             {chosenNode, "temp"},
00414             {chosenNode - 1, "current"}
00415         };
00416         event.colorNodes.push_back(chosenNode);
00417         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00418         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00419         event.indexBackArrow = {chosenNode - 1, 0};
00420         event.lines = {9};
00421
00422         events.emplace_back(event);
00423
00424         event.reset();

```

```

00425         event.titleNodes.emplace_back(0, "head");
00426         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00427         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00428         event.indexBackArrow = {chosenNode - 1, 0};
00429         event.lines = {10};
00430
00431         events.emplace_back(event);
00432     }
00433 }
00434
00435 return events;
00436 }
00437
00438 std::vector<EventAnimation> CLLScene::updateModeEvents(int chosenNode) {
00439     this->linkedList->resetEvents();
00440     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00441         return {};
00442
00443     this->linkedList->initHighlighter(
00444         constants::Highlighter::SLL::CODES_PATH[2].second,
00445         constants::Highlighter::SLL::CODES_PATH[2].first
00446     );
00447
00448     std::vector<EventAnimation> events;
00449     EventAnimation event;
00450
00451     event.titleNodes.emplace_back(0, "head|current");
00452     event.colorNodes.push_back(0);
00453     event.isPrintPreVal = true;
00454     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00455     event.lines = {0};
00456
00457     events.emplace_back(event);
00458
00459     if (chosenNode) {
00460         for (int i = 0; i <= chosenNode; ++i) {
00461             event.reset();
00462             event.titleNodes = {
00463                 {0, "head"},
00464                 {i, "current"}
00465             };
00466             event.colorNodes.push_back(i);
00467             event.isPrintPreVal = true;
00468             event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00469             event.lines = {1};
00470
00471             events.emplace_back(event);
00472
00473             if (i == chosenNode) break;
00474
00475             event.reset();
00476             event.titleNodes = {
00477                 {0, "head"},
00478                 {i, "current"}
00479             };
00480             event.colorNodes.push_back(i);
00481             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00482             event.isPrintPreVal = true;
00483             event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00484             event.lines = {2};
00485
00486             events.emplace_back(event);
00487         }
00488     }
00489
00490     event.reset();
00491     if (chosenNode == 0)
00492         event.titleNodes.emplace_back(0, "head|current");
00493     else
00494         event.titleNodes = {
00495             {0, "head"},
00496             {chosenNode, "current"}
00497         };
00498     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00499     event.lines = {3};
00500
00501     events.emplace_back(event);
00502
00503     return events;
00504 }
00505
00506 std::vector<EventAnimation> CLLScene::searchModeEvents(int chosenNode) {
00507     this->linkedList->resetEvents();
00508     this->linkedList->initHighlighter(
00509         constants::Highlighter::SLL::CODES_PATH[3].second,
00510         constants::Highlighter::SLL::CODES_PATH[3].first
00511     );

```

```

00512
00513     std::vector<EventAnimation> events;
00514     EventAnimation event;
00515
00516     event.titleNodes.emplace_back(0, "head|current");
00517     event.colorNodes.push_back(0);
00518     event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00519     event.lines = {0};
00520
00521     events.emplace_back(event);
00522
00523     for (int i = 0; i <= chosenNode; ++i) {
00524         if (i == chosenNode && chosenNode == this->linkedList->getSize())
00525             break;
00526
00527         event.reset();
00528         event.titleNodes = {
00529             {0, "head"},
00530             {i, "current"}
00531         };
00532         event.colorNodes.push_back(i);
00533         event.indexBackArrow = {this->linkedList->getSize() - 1, 0};
00534         event.lines = {1};
00535
00536         events.emplace_back(event);
00537
00538         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 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 28/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_CLLSCENE_HPP
00006 #define VISUALGO_CS162_CLLSCENE_HPP
00007
00008 #include "BaseScene.hpp"
00009 #include "MenuLinkedList.hpp"
00010 #include "core/LinkedList.hpp"
00011
00012 class CLLScene : public BaseScene{
00013 private:
00014     MenuLinkedList* menu;
00015     LinkedList* linkedList;
00016
00017     void init();
00018
00019 public:
00020     explicit CLLScene(sf::RenderWindow* window);
00021
00022     void reset();
00023
00024     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00025     void update() override;
00026     void render() override;
00027
00028     std::vector<EventAnimation> addModeEvents(int chosenNode);
00029     std::vector<EventAnimation> deleteModeEvents(int chosenNode);
00030     std::vector<EventAnimation> updateModeEvents(int chosenNode);
00031     std::vector<EventAnimation> searchModeEvents(int chosenNode);
00032 };
00033
00034 #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.](#)

```

00001 //
00002 // Created by dirii on 14/04/2023.
00003 //
00004
00005 #include "ControlMenu.hpp"
00006
00007 ControlMenu::ControlMenu(sf::RenderWindow *window) {
00008     this->window = window;
00009
00010     for (int i = 0; i < constants::ControlMenu::BUTTON_COUNT; ++i) {
00011         buttons[i] = new Button(
00012             this->window,
00013             constants::ControlMenu::buttonPos[i],
00014             constants::ControlMenu::buttonSize,
00015             constants::ControlMenu::BUTTON_NAMES[i],
00016             constants::ControlMenu::BUTTON_NAMES[i],
00017             constants::ControlMenu::BUTTON_NAME_SIZE,
00018             sf::Color::Black,
00019             constants::normalGray,
00020             constants::hoverGray,
00021             constants::clickGray
00022         );
00023     }
00024
00025     this->font.loadFromFile(constants::fontPath);
00026     this->textSpeed.setFont(font);

```

```

00027     this->textSpeed.setString(to_string_with_precision(this->speed));
00028     this->textSpeed.setCharacterSize(constants::ControlMenu::TEXT_SIZE);
00029     this->textSpeed.setFillColor(sf::Color::Black);
00030     this->textSpeed.setOrigin(
00031         this->textSpeed.getLocalBounds().width / 2.0f,
00032         this->textSpeed.getLocalBounds().height / 2.0f
00033     );
00034     this->textSpeed.setPosition(
00035         constants::ControlMenu::buttonPos[3].x + constants::ControlMenu::buttonSize.x * 2,
00036         constants::ControlMenu::buttonPos[3].y + constants::ControlMenu::buttonSize.y / 2.0f
00037     );
00038
00039     this->status = StatusCode::None;
00040     this->speed = 1;
00041 }
00042
00043 void ControlMenu::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
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 }
00074
00075 void ControlMenu::update() {
00076     for (auto &button : buttons) {
00077         button->update();
00078     }
00079     this->textSpeed.setString(to_string_with_precision(this->speed));
00080 }
00081
00082 void ControlMenu::render() {
00083     for (auto &button : buttons) {
00084         button->render();
00085     }
00086     this->window->draw(this->textSpeed);
00087 }
00088
00089 ControlMenu::StatusCode ControlMenu::getStatus() {
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 }
00095
00096 float ControlMenu::getSpeed() const {
00097     return this->speed;
00098 }
00099
00100 void ControlMenu::reset() {
00101     this->status = StatusCode::None;
00102 }

```

## 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.](#)

```
00001 //
00002 // Created by dirii on 14/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_CONTROLMENU_HPP
00006 #define VISUALGO_CS162_CONTROLMENU_HPP
00007
00008 #include <SFML/Graphics.hpp>
00009 #include "stuff/button.hpp"
00010 #include "stuff/ToStringWithPrecision.hpp"
00011 #include "Constants.hpp"
00012
00013 class ControlMenu {
00014 private:
00015     sf::RenderWindow* window;
00016
00017     Button* buttons[constants::ControlMenu::BUTTON_COUNT];
00018     sf::Font font;
00019     sf::Text textSpeed;
00020     float speed;
00021
00022 public:
00023     enum class StatusCode {
00024         PREVIOUS,
00025         PAUSE,
00026         PLAY,
00027         NEXT,
00028         None
00029     } status;
00030
00031     explicit ControlMenu(sf::RenderWindow* window);
00032     ~ControlMenu() = default;
00033
00034     void pollEvents(sf::Event event, sf::Vector2f mousePosView);
00035     void update();
00036     void render();
00037     void reset();
00038
00039     ControlMenu::StatusCode getStatus();
00040     [[nodiscard]] float getSpeed() const;
00041 };
00042
00043 #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.](#)

```

00001 //
00002 // Created by dirii on 27/03/2023.
00003 //
00004
00005 #include "DLLScene.hpp"
00006
00007 DLLScene::DLLScene(sf::RenderWindow *window) : BaseScene(window) {
00008     this->init();
00009 }
00010
00011 void DLLScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00012     if (this->isMenuOpen)
00013         this->menu->pollEvents(event, mousePosView);
00014
00015     this->controlMenu->pollEvents(event, mousePosView);
00016 }
00017
00018 void DLLScene::update() {
00019     if (this->isMenuOpen) {
00020         this->menu->update();
00021
00022         constants::MenuLinkedList::Button status = this->menu->getActiveOptionsMenu();
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 ==
00035 constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
00036                     if (this->menu->createModeValue[1] == "None")
00037                         break;
00038                     std::vector<std::string> values;
00039                     std::string value = this->menu->createModeValue[1];
00040                     std::stringstream ss(value);
00041                     std::string token;
00042                     while (std::getline(ss, token, ',')) {
00043                         values.push_back(token);
00044                     }
00045                     this->linkedList->createLinkedList(values);
00046                 } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
00047                     if (this->menu->createModeValue[2] == "None")
00048                         break;
00049                     std::vector<std::string> values;
00050                     std::string value = this->menu->createModeValue[2];
00051                     std::stringstream ss(value);
00052                     std::string token;
00053                     while (std::getline(ss, token, ','))
00054                         values.push_back(token);
00055                     this->linkedList->createLinkedList(values);
00056                     this->menu->createModeValue[2] = "None";
00057                 }
00058                 this->controlMenu->reset();
00059                 break;
00060             case constants::MenuLinkedList::Button::ADD_BUTTON:
00061                 if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
00062 this->menu->addModeValue[0].empty())
00063                     break;
00064                 this->linkedList->addNode(
00065                     std::stoi(this->menu->addModeValue[0]),
00066                     this->menu->addModeValue[1],
00067                     this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00068                 );
00069                 std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
00070 << std::endl;
00071                 this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00072                 this->controlMenu->reset();
00073                 break;
00074             case constants::MenuLinkedList::Button::DELETE_BUTTON:
00075                 if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
00076                     break;
00077                 this->linkedList->deleteNode(
00078                     std::stoi(this->menu->deleteModeValue),
00079                     this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))

```

```

00080         );
00081
00082         std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00083         this->menu->deleteModeValue = "None";
00084         this->controlMenu->reset();
00085         break;
00086     case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00087         if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
"None" || this->menu->updateModeValue[0].empty())
00088             break;
00089
00090         this->linkedList->updateNode(
00091             std::stoi(this->menu->updateModeValue[0]),
00092             this->menu->updateModeValue[1],
00093             this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00094         );
00095
00096         std::cout << "Update: " << this->menu->updateModeValue[0] << " " <<
this->menu->updateModeValue[1] << std::endl;
00097         this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00098         this->controlMenu->reset();
00099         break;
00100     case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00101         if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00102             break;
00103
00104         this->linkedList->searchNode(
00105             this->searchModeEvents(this->linkedList->findValue(this->menu->searchModeValue))
00106         );
00107
00108         std::cout << "Search: " << this->menu->searchModeValue << std::endl;
00109         this->menu->searchModeValue = "None";
00110         this->controlMenu->reset();
00111         break;
00112     }
00113 }
00114
00115 this->controlMenu->update();
00116
00117 this->linkedList->processControlMenu(this->controlMenu->getStatus());
00118 this->linkedList->setSpeed(this->controlMenu->getSpeed());
00119
00120 this->linkedList->update();
00121 }
00122
00123 void DLLScene::render() {
00124     if (this->isMenuOpen)
00125         this->menu->render();
00126
00127     if (this->isDemoCodeOpen)
00128         this->linkedList->renderHighlighter();
00129
00130     this->controlMenu->render();
00131     this->linkedList->render();
00132 }
00133
00134 void DLLScene::init() {
00135     this->menu = new MenuLinkedList(this->window);
00136     this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::DOUBLY);
00137 }
00138
00139 void DLLScene::reset() {
00140     this->menu->resetActiveOptionsMenu();
00141 }
00142
00143 std::vector<EventAnimation> DLLScene::addModeEvents(int chosenNode) {
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     }
00221     else {
00222         event.titleNodes.emplace_back(chosenNode, "head|tail");
00223         event.lines = {5, 6, 7};
00224     }
00225     event.colorNodes = {chosenNode};
00226
00227     events.emplace_back(event);
00228 }
00229 else if (chosenNode == size) {
00230     this->linkedList->initHighlighter(
00231         constants::Highlighter::DLL::CODES_PATH[1].second,
00232         constants::Highlighter::DLL::CODES_PATH[1].first
00233     );
00234
00235     event.titleNodes.emplace_back(chosenNode, "temp");
00236     if (size == 1)
00237         event.titleNodes.emplace_back(0, "head|tail");
00238     else if (size > 1){
00239         event.titleNodes.emplace_back(0, "head");
00240         event.titleNodes.emplace_back(size - 1, "tail");
00241     }
00242     event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
00243     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00244     event.colorNodes.emplace_back(chosenNode);
00245     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00246     event.lines = {0, 1};
00247
00248     events.emplace_back(event);
00249
00250     event.reset();

```

```

00251
00252     event.titleNodes.emplace_back(chosenNode, "temp");
00253     if (size == 1)
00254         event.titleNodes.emplace_back(0, "head|tail");
00255     else if (size > 1){
00256         event.titleNodes.emplace_back(0, "head");
00257         event.titleNodes.emplace_back(size - 1, "tail");
00258     }
00259     event.hiddenArrows.emplace_back(size - 1, NodeInfo::ArrowType::RIGHT);
00260     event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00261     event.colorNodes.emplace_back(chosenNode);
00262     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00263     event.lines = {2};
00264
00265     events.emplace_back(event);
00266
00267     event.reset();
00268     event.titleNodes.emplace_back(chosenNode, "temp");
00269     if (size == 1)
00270         event.titleNodes.emplace_back(0, "head|tail");
00271     else if (size > 1){
00272         event.titleNodes.emplace_back(0, "head");
00273         event.titleNodes.emplace_back(size - 1, "tail");
00274     }
00275     event.colorArrows = {
00276         {chosenNode, NodeInfo::ArrowType::LEFT},
00277         {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
00278     };
00279     event.colorNodes.emplace_back(chosenNode - 1);
00280     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00281     event.lines = {3};
00282
00283     events.emplace_back(event);
00284
00285     event.reset();
00286     event.titleNodes = {
00287         {chosenNode, "tail"},
00288         {0, "head"}
00289     };
00290     event.colorNodes = {chosenNode};
00291     event.lines = {4};
00292
00293     events.emplace_back(event);
00294 }
00295 else {
00296     this->linkedList->initHighlighter(
00297         constants::Highlighter::DLL::CODES_PATH[2].second,
00298         constants::Highlighter::DLL::CODES_PATH[2].first
00299     );
00300
00301     event.titleNodes = {
00302         {chosenNode, "temp"},
00303         {0, "head"},
00304         {size, "tail"}
00305     };
00306     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00307     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00308     event.colorNodes.emplace_back(chosenNode);
00309     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00310     event.lines = {0, 1};
00311
00312     events.emplace_back(event);
00313
00314     event.reset();
00315     event.titleNodes = {
00316         {chosenNode, "temp"},
00317         {0, "head|current"},
00318         {size, "tail"}
00319     };
00320     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00321     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00322     event.colorNodes.emplace_back(0);
00323     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00324     event.lines = {2};
00325
00326     events.emplace_back(event);
00327
00328     for (int i = 0; i < chosenNode; ++i) {
00329         event.reset();
00330         event.titleNodes = {
00331             {chosenNode, "temp"},
00332             {0, "head"},
00333             {size, "tail"},
00334             {i, "current"}
00335         };
00336         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00337         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);

```

```

00338         event.colorNodes.emplace_back(i);
00339         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00340         event.lines = {3};
00341
00342         events.emplace_back(event);
00343
00344         if (i == chosenNode - 1)
00345             break;
00346
00347         event.reset();
00348         event.titleNodes = {
00349             {chosenNode, "temp"},
00350             {0, "head"},
00351             {size, "tail"},
00352             {i, "current"}
00353         };
00354         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00355         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::LEFT);
00356         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00357         // event.colorArrows.emplace_back(i + 1 + (i + 1 == chosenNode),
NodeInfo::ArrowType::LEFT);
00358         event.colorNodes.emplace_back(i);
00359         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00360         event.lines = {4};
00361
00362         events.emplace_back(event);
00363     }
00364
00365     event.reset();
00366     event.titleNodes = {
00367         {chosenNode, "temp"},
00368         {0, "head"},
00369         {size, "tail"},
00370         {chosenNode - 1, "current"}
00371     };
00372     event.colorArrows = {
00373         {chosenNode, NodeInfo::ArrowType::RIGHT},
00374         {chosenNode, NodeInfo::ArrowType::LEFT}
00375     };
00376     event.colorNodes.emplace_back(chosenNode);
00377     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00378     event.isPrintNormal = true;
00379     event.lines = {5, 6};
00380
00381     events.emplace_back(event);
00382
00383     event.reset();
00384     event.titleNodes = {
00385         {chosenNode, "temp"},
00386         {0, "head"},
00387         {size, "tail"}
00388     };
00389     event.colorNodes.emplace_back(chosenNode);
00390     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00391     event.lines = {7, 8};
00392
00393     events.emplace_back(event);
00394 }
00395
00396 return events;
00397 }
00398
00399 std::vector<EventAnimation> DLLScene::deleteModeEvents(int chosenNode) {
00400     this->linkedList->resetEvents();
00401     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00402         return {};
00403
00404     std::vector<EventAnimation> events;
00405     EventAnimation event;
00406     int size = this->linkedList->getSize();
00407
00408     if (chosenNode == 0) {
00409         this->linkedList->initHighlighter(
00410             constants::Highlighter::DLL::CODES_PATH[3].second,
00411             constants::Highlighter::DLL::CODES_PATH[3].first
00412         );
00413
00414         if (size == 1) {
00415             event.titleNodes.emplace_back(chosenNode, "head|tail|temp");
00416             event.colorNodes.emplace_back(chosenNode);
00417             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00418             event.lines = {0, 1, 2};
00419
00420             events.emplace_back(event);
00421
00422             event.reset();
00423             event.statusChosenNode = NodeInfo::StatusNode::Visible;

```

```

00424         event.lines = {5, 6, 7};
00425
00426         events.emplace_back(event);
00427     }
00428     else {
00429         event.titleNodes = {
00430             {chosenNode, "head|temp"},
00431             {size - 1, "tail"}
00432         };
00433         event.colorNodes.emplace_back(chosenNode);
00434         event.lines = {0, 1};
00435
00436         events.emplace_back(event);
00437
00438         event.reset();
00439         if (size == 2)
00440             event.titleNodes.emplace_back(size - 1, "head|tail");
00441         else
00442             event.titleNodes = {
00443                 {size - 1, "tail"},
00444                 {chosenNode + 1, "head" }
00445             };
00446         event.titleNodes.emplace_back(chosenNode, "temp");
00447         event.colorNodes.emplace_back(chosenNode + 1);
00448         // event.isPrintNormal = true;
00449         // event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00450         event.lines = {2};
00451
00452         events.emplace_back(event);
00453
00454         event.reset();
00455         if (size == 2)
00456             event.titleNodes.emplace_back(size - 1, "head|tail");
00457         else
00458             event.titleNodes = {
00459                 {size - 1, "tail"},
00460                 {chosenNode + 1, "head" }
00461             };
00462         event.titleNodes.emplace_back(chosenNode, "temp");
00463         event.colorNodes.emplace_back(chosenNode);
00464         event.hiddenArrows = {
00465             // {chosenNode, NodeInfo::ArrowType::RIGHT},
00466             {chosenNode + 1, NodeInfo::ArrowType::LEFT}
00467         };
00468         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00469         event.isPrintNormal = true;
00470         event.lines = {3, 4};
00471
00472         events.emplace_back(event);
00473
00474         event.reset();
00475         if (size == 2)
00476             event.titleNodes.emplace_back(size - 1, "head|tail");
00477         else
00478             event.titleNodes = {
00479                 {size - 1, "tail"},
00480                 {chosenNode + 1, "head" }
00481             };
00482         event.hiddenArrows = {
00483             {chosenNode + 1, NodeInfo::ArrowType::LEFT}
00484         };
00485         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00486         event.lines = {7};
00487
00488         events.emplace_back(event);
00489     }
00490 }
00491 else if (chosenNode == size - 1) {
00492     this->linkedList->initHighlighter(
00493         constants::Highlighter::DLL::CODES_PATH[4].second,
00494         constants::Highlighter::DLL::CODES_PATH[4].first
00495     );
00496
00497     event.titleNodes = {
00498         {0, "head"},
00499         {chosenNode, "tail|temp"}
00500     };
00501     event.colorNodes.emplace_back(chosenNode);
00502     event.lines = {0, 1};
00503
00504     events.emplace_back(event);
00505
00506     event.reset();
00507     if (size == 2)
00508         event.titleNodes.emplace_back(0, "head|tail");
00509     else
00510         event.titleNodes = {

```

```

00511         {chosenNode - 1, "tail"},
00512         {0, "head" }
00513     };
00514     event.titleNodes.emplace_back(chosenNode, "temp");
00515     event.colorNodes.emplace_back(chosenNode - 1);
00516     event.lines = {2};
00517
00518     events.emplace_back(event);
00519
00520     event.reset();
00521     if (size == 2)
00522         event.titleNodes.emplace_back(0, "head|tail");
00523     else
00524         event.titleNodes = {
00525             {chosenNode - 1, "tail"},
00526             {0, "head" }
00527         };
00528     event.titleNodes.emplace_back(chosenNode, "temp");
00529     event.colorNodes.emplace_back(chosenNode);
00530     event.hiddenArrows = {
00531         {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
00532     };
00533     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00534     event.isPrintNormal = true;
00535     event.lines = {3};
00536
00537     events.emplace_back(event);
00538
00539     event.reset();
00540     if (size == 2)
00541         event.titleNodes.emplace_back(0, "head|tail");
00542     else
00543         event.titleNodes = {
00544             {chosenNode - 1, "tail"},
00545             {0, "head" }
00546         };
00547     event.hiddenArrows = {
00548         {chosenNode - 1, NodeInfo::ArrowType::RIGHT}
00549     };
00550     event.statusChosenNode = NodeInfo::StatusNode::Visible;
00551     event.lines = {4};
00552
00553     events.emplace_back(event);
00554 }
00555 else {
00556     this->linkedList->initHighlighter(
00557         constants::Highlighter::DLL::CODES_PATH[5].second,
00558         constants::Highlighter::DLL::CODES_PATH[5].first
00559     );
00560
00561     event.titleNodes = {
00562         {0, "head|temp"},
00563         {size - 1, "tail"}
00564     };
00565     event.colorNodes.emplace_back(0);
00566     event.lines = {0, 1};
00567
00568     events.emplace_back(event);
00569
00570     for (int i = 0; i <= chosenNode; ++i) {
00571         event.reset();
00572         event.titleNodes = {
00573             {0, "head"},
00574             {i, "temp"},
00575             {size - 1, "tail"}
00576         };
00577         event.colorNodes.emplace_back(i);
00578         event.lines = {2};
00579
00580         events.emplace_back(event);
00581
00582         if (i == chosenNode)
00583             break;
00584
00585         event.reset();
00586         event.titleNodes = {
00587             {0, "head"},
00588             {i, "temp"},
00589             {size - 1, "tail"}
00590         };
00591         event.colorNodes.emplace_back(i);
00592         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00593         event.lines = {3};
00594
00595         events.emplace_back(event);
00596     }
00597

```



```

00598     event.reset();
00599     event.titleNodes = {
00600         {0, "head"},
00601         {chosenNode, "temp"},
00602         {size - 1, "tail"}
00603     };
00604     event.colorNodes.emplace_back(chosenNode);
00605     event.colorArrows = {
00606         {chosenNode - 1, NodeInfo::ArrowType::RIGHT},
00607         {chosenNode + 1, NodeInfo::ArrowType::LEFT}
00608     };
00609     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00610     event.isPrintNormal = true;
00611     event.lines = {4, 5};
00612
00613     events.emplace_back(event);
00614
00615     event.reset();
00616     event.titleNodes = {
00617         {0, "head"},
00618         {size - 1, "tail"}
00619     };
00620     event.statusChosenNode = NodeInfo::StatusNode::Visible;
00621     event.lines = {6};
00622
00623     events.emplace_back(event);
00624 }
00625
00626 return events;
00627 }
00628
00629 std::vector<EventAnimation> DLLScene::updateModeEvents(int chosenNode) {
00630     this->linkedList->resetEvents();
00631     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00632         return {};
00633
00634     this->linkedList->initHighlighter(
00635         constants::Highlighter::DLL::CODES_PATH[6].second,
00636         constants::Highlighter::DLL::CODES_PATH[6].first
00637     );
00638
00639     std::vector<EventAnimation> events;
00640     EventAnimation event;
00641     int size = this->linkedList->getSize();
00642
00643     if (size > 1)
00644         event.titleNodes = {
00645             {0, "head|current"},
00646             {size - 1, "tail"}
00647         };
00648     else
00649         event.titleNodes = {
00650             {0, "head|tail|current"}
00651         };
00652     event.colorNodes.push_back(0);
00653     event.isPrintPreVal = true;
00654     event.lines = {0};
00655
00656     events.emplace_back(event);
00657
00658     if (chosenNode) {
00659         for (int i = 0; i <= chosenNode; ++i) {
00660             event.reset();
00661             event.titleNodes = {
00662                 {0, "head"},
00663                 {size - 1, "tail"},
00664                 {i, "current"},
00665             };
00666             event.colorNodes.push_back(i);
00667             event.isPrintPreVal = true;
00668             event.lines = {1};
00669
00670             events.emplace_back(event);
00671
00672             if (i == chosenNode) break;
00673
00674             event.reset();
00675             event.titleNodes = {
00676                 {0, "head"},
00677                 {i, "current"},
00678                 {size - 1, "tail"}
00679             };
00680             event.colorNodes.push_back(i);
00681             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00682             event.isPrintPreVal = true;
00683             event.lines = {2};
00684

```

```

00685         events.emplace_back(event);
00686     }
00687 }
00688
00689 event.reset();
00690 if (size == 1)
00691     event.titleNodes = {
00692         {0, "head|tail|current"}
00693     };
00694 else if (chosenNode == size - 1)
00695     event.titleNodes = {
00696         {0, "head"},
00697         {chosenNode, "current|tail"}
00698     };
00699 else if (chosenNode == 0)
00700     event.titleNodes = {
00701         {0, "head|current"},
00702         {size - 1, "tail"}
00703     };
00704 else
00705     event.titleNodes = {
00706         {0, "head"},
00707         {chosenNode, "current"},
00708         {size - 1, "tail"}
00709     };
00710 event.lines = {3};
00711
00712 events.emplace_back(event);
00713
00714 return events;
00715 }
00716
00717 std::vector<EventAnimation> DLLScene::searchModeEvents(int chosenNode) {
00718     this->linkedList->resetEvents();
00719     this->linkedList->initHighlighter(
00720         constants::Highlighter::DLL::CODES_PATH[7].second,
00721         constants::Highlighter::DLL::CODES_PATH[7].first
00722     );
00723
00724     std::vector<EventAnimation> events;
00725     EventAnimation event;
00726     int size = this->linkedList->getSize();
00727
00728     if (size > 1)
00729         event.titleNodes = {
00730             {0, "head|current"},
00731             {size - 1, "tail"}
00732         };
00733     else
00734         event.titleNodes = {
00735             {0, "head|tail|current"}
00736         };
00737     event.colorNodes.push_back(0);
00738     event.lines = {0};
00739
00740     events.emplace_back(event);
00741
00742     for (int i = 0; i <= chosenNode; ++i) {
00743         if (i == chosenNode && chosenNode == this->linkedList->getSize())
00744             break;
00745
00746         event.reset();
00747         event.titleNodes = {
00748             {0, "head"},
00749             {size - 1, "tail"},
00750             {i, "current"}
00751         };
00752         event.colorNodes.push_back(i);
00753         event.lines = {1};
00754
00755         events.emplace_back(event);
00756
00757         if (i == chosenNode) break;
00758
00759         event.reset();
00760         event.titleNodes = {
00761             {0, "head"},
00762             {size - 1, "tail"},
00763             {i, "current"}
00764         };
00765         event.colorNodes.push_back(i);
00766         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00767         event.lines = {4};
00768
00769         events.emplace_back(event);
00770     }
00771 }

```

```

00772     if (chosenNode == this->linkedList->getSize()) {
00773         event.reset();
00774         event.titleNodes.emplace_back(0, "head");
00775         event.titleNodes.emplace_back(size - 1, "tail");
00776         event.lines = {5};
00777
00778         events.emplace_back(event);
00779     } else {
00780         event.reset();
00781         event.titleNodes = {
00782             {0, "head"},
00783             {size - 1, "tail"},
00784             {chosenNode, "current"}
00785         };
00786         event.colorNodes.push_back(chosenNode);
00787         event.lines = {2, 3};
00788
00789         events.emplace_back(event);
00790     }
00791
00792     return events;
00793 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 27/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_DLLSCENE_HPP
00006 #define VISUALGO_CS162_DLLSCENE_HPP
00007
00008 #include "BaseScene.hpp"
00009 #include "MenuLinkedList.hpp"
00010 #include "core/LinkedList.hpp"
00011
00012 class DLLScene : public BaseScene {
00013 private:
00014     MenuLinkedList* menu;
00015     LinkedList* linkedList;
00016
00017     void init();
00018
00019 public:
00020     explicit DLLScene(sf::RenderWindow* window);
00021
00022     void reset();
00023
00024     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00025     void update() override;
00026     void render() override;
00027
00028     std::vector<EventAnimation> addModeEvents(int chosenNode);
00029     std::vector<EventAnimation> deleteModeEvents(int chosenNode);
00030     std::vector<EventAnimation> updateModeEvents(int chosenNode);
00031     std::vector<EventAnimation> searchModeEvents(int chosenNode);
00032 };
00033
00034 #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.](#)

```
00001 //
00002 // Created by dirii on 27/03/2023.
00003 //
00004
00005 #include "DynamicArrayScene.hpp"
00006
00007 DynamicArrayScene::DynamicArrayScene(sf::RenderWindow *window) : BaseScene(window) {
00008     this->init();
00009 }
00010
00011 void DynamicArrayScene::update() {
00012     if (this->isMenuOpen) {
00013         this->menu->update();
00014
00015         constants::MenuArray::Button status = this->menu->getActiveOptionsMenu();
00016         constants::MenuArray::CreateMode::Button createMode;
00017         switch (status){
00018             case constants::MenuArray::Button::CREATE_BUTTON:
00019                 createMode = this->menu->getActiveCreateMode();
00020                 if (createMode == constants::MenuArray::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->array->createArray(size);
00027                 } else if (createMode ==
00028                     constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
00029                     if (this->menu->createModeValue[1] == "None")
00030                         break;
00031                     std::vector<std::string> values;
00032                     std::string value = this->menu->createModeValue[1];
00033                     std::stringstream ss(value);
00034                     std::string token;
00035                     while (std::getline(ss, token, ',')) {
00036                         values.push_back(token);
00037                     }
00038                     this->array->createArray(values);
00039                 } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
00040                     if (this->menu->createModeValue[2] == "None")
00041                         break;
00042                     std::vector<std::string> values;
00043                     std::string value = this->menu->createModeValue[2];
00044                     std::stringstream ss(value);
00045                     std::string token;
00046                     while (std::getline(ss, token, ','))
00047                         values.push_back(token);
00048                     this->array->createArray(values);
00049                     this->menu->createModeValue[2] = "None";
00050                 }
00051                 this->controlMenu->reset();
00052                 break;
00053             case constants::MenuArray::Button::ADD_BUTTON:
00054                 if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
00055                     this->menu->addModeValue[0].empty())
00056                     break;
00057                 this->array->addSquare(
00058                     std::stoi(this->menu->addModeValue[0]),
00059                     this->menu->addModeValue[1],
00060                     this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00061                 );
00062                 std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
00063                 << std::endl;
00064                 this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00065                 this->controlMenu->reset();
00066                 break;
00067             case constants::MenuArray::Button::DELETE_BUTTON:
00068                 if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
```

```

00068         break;
00069
00070         this->array->deleteSquare(
00071             std::stoi(this->menu->deleteModeValue),
00072             this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
00073         );
00074
00075         std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00076         this->menu->deleteModeValue = "None";
00077         this->controlMenu->reset();
00078         break;
00079     case constants::MenuArray::Button::UPDATE_BUTTON:
00080         if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
00081             "None" || this->menu->updateModeValue[0].empty())
00082             break;
00083
00084         this->array->updateSquare(
00085             std::stoi(this->menu->updateModeValue[0]),
00086             this->menu->updateModeValue[1],
00087             this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00088         );
00089
00090         std::cout << "Update: " << this->menu->updateModeValue[0] << " " <<
00091             this->menu->updateModeValue[1] << std::endl;
00092         this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00093         this->controlMenu->reset();
00094         break;
00095     case constants::MenuArray::Button::SEARCH_BUTTON:
00096         if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00097             break;
00098
00099         this->array->searchSquare(
00100             this->searchModeEvents(this->array->findValue(this->menu->searchModeValue))
00101         );
00102
00103         std::cout << "Search: " << this->menu->searchModeValue << std::endl;
00104         this->menu->searchModeValue = "None";
00105         this->controlMenu->reset();
00106         break;
00107     case constants::MenuArray::Button::ALLOCATE_BUTTON:
00108         if (this->menu->allocateModeValue == "None" || this->menu->allocateModeValue.empty())
00109             break;
00110
00111         this->array->allocateSquare(
00112             std::stoi(this->menu->allocateModeValue),
00113             this->allocateModeEvents(std::stoi(this->menu->allocateModeValue))
00114         );
00115
00116         std::cout << "Allocate: " << this->menu->allocateModeValue << std::endl;
00117         this->menu->allocateModeValue = "None";
00118         this->controlMenu->reset();
00119         break;
00120     }
00121
00122     this->controlMenu->update();
00123
00124     this->array->processControlMenu(this->controlMenu->getStatus());
00125     this->array->setSpeed(this->controlMenu->getSpeed());
00126
00127     this->array->update();
00128 }
00129 void DynamicArrayScene::render() {
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 }
00139
00140 void DynamicArrayScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00141     if (this->isMenuOpen)
00142         this->menu->pollEvents(event, mousePosView);
00143
00144     this->controlMenu->pollEvents(event, mousePosView);
00145 }
00146
00147 void DynamicArrayScene::init() {
00148     this->menu = new MenuArray(this->window, constants::MenuArray::Type::DYNAMIC);
00149     this->array = new Array(this->window, Array::TypeArray::DYNAMIC);
00150 }
00151
00152 void DynamicArrayScene::reset() {

```

```

00153     this->menu->resetActiveOptionsMenu();
00154 }
00155
00156 std::vector<EventAnimation> DynamicArrayScene::addModeEvents(int chosenNode) {
00157     this->array->resetEvents();
00158     if (chosenNode < 0 || chosenNode > this->array->getSize())
00159         return {};
00160
00161     // init highlighter
00162     // ...
00163
00164     int size = this->array->getSize() + 1,
00165         squaresSize = this->array->getSquaresSize();
00166     std::vector<EventAnimation> events;
00167     EventAnimation event;
00168
00169     if (size > squaresSize) {
00170         ++squaresSize;
00171         event.eventSquares.assign(squaresSize, EventSquare());
00172         event.eventSquaresTemp.assign(squaresSize, EventSquare());
00173         for (auto &square : event.eventSquares) {
00174             square.status = Square::Status::active;
00175             square.isPrintPreVal = true;
00176         }
00177         event.eventSquares.back().status = Square::Status::hidden;
00178         if (size > 1)
00179             event.eventSquares[size - 2].title = "n";
00180         for (auto &square : event.eventSquaresTemp) {
00181             square.status = Square::Status::inactive;
00182             square.isPrintPreVal = true;
00183         }
00184
00185         events.emplace_back(event);
00186
00187         for (int i = 0; i < size - 1; ++i) {
00188             event = EventAnimation();
00189             event.eventSquares.assign(squaresSize, EventSquare());
00190             event.eventSquaresTemp.assign(squaresSize, EventSquare());
00191             for (auto &square : event.eventSquares) {
00192                 square.status = Square::Status::active;
00193                 square.isPrintPreVal = true;
00194             }
00195             event.eventSquares.back().status = Square::Status::hidden;
00196             if (size > 1)
00197                 event.eventSquares[size - 2].title = "n";
00198             for (auto &square : event.eventSquaresTemp) {
00199                 square.status = Square::Status::inactive;
00200                 square.isPrintPreVal = true;
00201             }
00202             for (int j = 0; j < i; ++j) {
00203                 event.eventSquaresTemp[j].status = Square::Status::active;
00204                 event.eventSquaresTemp[j].isPrintPreVal = false;
00205             }
00206             event.eventSquaresTemp[i].status = Square::Status::chosen;
00207             event.eventSquaresTemp[i].title = "m";
00208
00209             events.emplace_back(event);
00210
00211             event.eventSquaresTemp[i].isPrintPreVal = false;
00212             event.eventSquares[i].status = Square::Status::chosen;
00213
00214             events.emplace_back(event);
00215         }
00216     }
00217
00218     event = EventAnimation();
00219     event.eventSquares.assign(squaresSize, EventSquare());
00220     event.eventSquaresTemp.assign(squaresSize, EventSquare());
00221     for (auto &square : event.eventSquares) {
00222         square.status = Square::Status::active;
00223         square.isPrintPreVal = true;
00224     }
00225     for (int i = size - 1; i < squaresSize; ++i)
00226         event.eventSquares[i].status = Square::Status::inactive;
00227     if (size > 1)
00228         event.eventSquares[size - 2].title = "n";
00229     for (auto &square : event.eventSquaresTemp) {
00230         square.status = Square::Status::hidden;
00231     }
00232
00233     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 }
00278
00279 std::vector<EventAnimation> DynamicArrayScene::deleteModeEvents(int chosenNode) {
00280     this->array->resetEvents();
00281     if (chosenNode < 0 || chosenNode >= this->array->getSize())
00282         return {};
00283
00284     // init highlighter
00285     // ...
00286
00287     int size = this->array->getSize(),
00288         squaresSize = this->array->getSquaresSize();
00289     std::vector<EventAnimation> events;
00290     EventAnimation event;
00291
00292     for (int i = chosenNode; i < size - 1; ++i) {
00293         event = EventAnimation();
00294         event.eventSquares.assign(squaresSize, EventSquare());
00295         for (auto &square : event.eventSquares) {
00296             square.status = Square::Status::active;
00297             square.isPrintPreVal = true;
00298         }
00299         for (int j = size; j < squaresSize; ++j)
00300             event.eventSquares[j].status = Square::Status::inactive;
00301         for (int j = 0; j < i; ++j)
00302             event.eventSquares[j].isPrintPreVal = false;
00303         event.eventSquares[i].status = Square::Status::chosen;
00304         for (auto &square : event.eventSquaresTemp)
00305             square.status = Square::Status::hidden;
00306         event.eventSquares[size - 1].title = "n";
00307
00308         events.emplace_back(event);
00309
00310         event.eventSquares[i].isPrintPreVal = false;
00311         event.eventSquares[i + 1].status = Square::Status::chosen;
00312
00313         events.emplace_back(event);
00314     }
00315
00316     event = EventAnimation();
00317     event.eventSquares.assign(squaresSize, EventSquare());
00318     for (int i = 0; i < size - 1; ++i) {
00319         event.eventSquares[i].status = Square::Status::active;
00320         if (i == size - 2)
00321             event.eventSquares[i].title = "n";
00322     }
00323     for (int i = size - 1; i < squaresSize; ++i)
00324         event.eventSquares[i].status = Square::Status::inactive;
00325
00326     events.emplace_back(event);

```

```

00327
00328     return events;
00329 }
00330
00331 std::vector<EventAnimation> DynamicArrayScene::updateModeEvents(int chosenNode) {
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
00356     events.emplace_back(event);
00357
00358     return events;
00359 }
00360
00361 std::vector<EventAnimation> DynamicArrayScene::searchModeEvents(int chosenNode) {
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 }
00401
00402 std::vector<EventAnimation> DynamicArrayScene::allocateModeEvents(int newSize) {
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 }

```

## 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.](#)

```
00001 //
00002 // Created by dirii on 27/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_DYNAMICARRAYSCENE_HPP
00006 #define VISUALGO_CS162_DYNAMICARRAYSCENE_HPP
00007
00008 #include "BaseScene.hpp"
00009 #include "MenuArray.hpp"
00010 #include "core/Array.hpp"
00011
00012 class DynamicArrayScene : public BaseScene{
00013 private:
00014     MenuArray* menu;
00015     Array* array;
00016
00017     void init();
00018
00019 public:
00020     explicit DynamicArrayScene(sf::RenderWindow* window);
00021
00022     void reset();
00023
00024     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00025     void update() override;
00026     void render() override;
00027
00028     std::vector<EventAnimation> addModeEvents(int chosenNode);
00029     std::vector<EventAnimation> deleteModeEvents(int chosenNode);
00030     std::vector<EventAnimation> updateModeEvents(int chosenNode);
00031     std::vector<EventAnimation> searchModeEvents(int chosenNode);
00032     std::vector<EventAnimation> allocateModeEvents(int newSize);
00033 };
00034
00035 #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.](#)

```
00001 //
00002 // Created by dirii on 15/04/2023.
00003 //
00004
00005 #include "Highlighter.hpp"
00006
00007 Highlighter::Highlighter(sf::RenderWindow *window, int linesCount, const char *codePath) {
00008     this->window = window;
00009     this->linesCount = linesCount;
00010
00011     this->codeTexture.loadFromFile(codePath);
00012     this->codeSprite.setTexture(this->codeTexture);
00013     this->codeSprite.setScale(constants::Highlighter::codeScale);
00014
00015     this->codeSprite.setOrigin(
00016         this->codeSprite.getLocalBounds().width,
00017         this->codeSprite.getLocalBounds().height
00018     );
00019
00020     this->codeSprite.setPosition(constants::Highlighter::codePos);
00021
00022     float heightTop = 43;
00023
00024     this->rectSize = sf::Vector2f(
00025         this->codeSprite.getGlobalBounds().width,
00026         ((this->codeSprite.getLocalBounds().height - heightTop * 2) /
00027         static_cast<float>(this->linesCount)) * constants::Highlighter::codeScale.y
00028     );
00029
00030     for (int i = 0; i < this->linesCount; ++i) {
00031         sf::RectangleShape rect(this->rectSize);
00032         rect.setOrigin(rect.getLocalBounds().width, rect.getLocalBounds().height);
00033         rect.setFillColor(constants::transparentGreen);
00034         rect.setPosition(
00035             this->codeSprite.getPosition().x,
00036             this->codeSprite.getPosition().y - (heightTop * constants::Highlighter::codeScale.y) -
00037             static_cast<float>(this->linesCount - 1 - i) * this->rectSize.y
00038         );
00039         this->lines.push_back(rect);
00040     }
00041
00042 void Highlighter::toggle(std::vector<int> linesList) {
00043     this->toggleLines = std::move(linesList);
00044 }
00045
00046 void Highlighter::render() {
00047     this->window->draw(this->codeSprite);
00048
00049     for (auto &i : this->toggleLines) {
00050         this->window->draw(this->lines[i]);
00051     }
00052 }
00053
00054 void Highlighter::resetToggle() {
00055     this->toggleLines.clear();
00056 }
```

## 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.](#)

```
00001 //
00002 // Created by dirii on 15/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_HIGHLIGHTER_HPP
00006 #define VISUALGO_CS162_HIGHLIGHTER_HPP
00007
00008 #include <SFML/Graphics.hpp>
00009 #include "Constants.hpp"
00010
00011 class Highlighter {
00012 private:
00013     sf::RenderWindow* window;
00014
00015     sf::Texture codeTexture;
00016     sf::Sprite codeSprite;
00017
00018     int linesCount;
00019
00020     std::vector<sf::RectangleShape> lines;
00021     std::vector<int> toggleLines;
00022
00023     sf::Vector2f rectSize;
00024
00025 public:
00026     Highlighter(sf::RenderWindow* window, int linesCount, const char* codePath);
00027
00028     void toggle(std::vector<int> lines);
00029     void resetToggle();
00030     void render();
00031 };
00032
00033 #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.](#)

```
00001 //
00002 // Created by dirii on 30/03/2023.
00003 //
00004
00005 #include "MainMenu.hpp"
00006
00007 MainMenu::MainMenu(sf::RenderWindow *window) : BaseScene(window) {
00008     this->modeButton = new Button;
00009 }
00010
00011 void MainMenu::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00012
00013 }
00014
00015 void MainMenu::update() {
00016
00017 }
00018
00019 void MainMenu::render() {
00020
00021 }
```

## 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.](#)

```
00001 //
00002 // Created by dirii on 30/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_MAINMENU_HPP
00006 #define VISUALGO_CS162_MAINMENU_HPP
00007
00008 #include "BaseScene.hpp"
00009
00010 class MainMenu : public BaseScene{
00011 public:
00012     explicit MainMenu(sf::RenderWindow* window);
00013
00014     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00015     void update() override;
00016     void render() override;
00017 };
00018
00019 #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.](#)

```
00001 //
00002 // Created by dirii on 01/05/2023.
00003 //
00004
00005 #include "MenuArray.hpp"
00006
00007 MenuArray::MenuArray(sf::RenderWindow *window, constants::MenuArray::Type _typeArray) {
00008     this->window = window;
00009     this->typeArray = _typeArray;
00010     this->init();
00011 }
00012
00013 void MenuArray::init() {
00014     this->initButtons();
00015     this->initCreateMode();
00016     this->initAddMode();
00017     this->initDeleteMode();
00018     this->initUpdateMode();
00019     this->initSearchMode();
00020     this->initAllocateMode();
00021
00022     this->activeOptionsMenu = constants::MenuArray::Button::NONE;
00023 }
00024
00025 void MenuArray::initButtons() {
00026     for (int i = 0; i < constants::MenuArray::BUTTON_COUNT; i++) {
00027         sf::Vector2f position = sf::Vector2f(
00028             constants::sideButtonSize.x + constants::distance2ModeButtons,
00029             constants::submenuButtonPos.y + (constants::optionButtonSize.y +
00030             constants::distance2ModeButtons / 10) * static_cast<float>(i)
00031         );
00032         if (i == constants::MenuArray::BUTTON_COUNT - 1 && this->typeArray ==
00033             constants::MenuArray::Type::STATIC)
```

```

00032         position = sf::Vector2f(-100, -100);
00033     this->buttons[i] = new Button(
00034         this->window,
00035         position,
00036         constants::optionButtonSize,
00037         constants::MenuArray::BUTTON_NAMES[i],
00038         constants::MenuArray::BUTTON_NAMES[i],
00039         constants::MenuArray::BUTTON_NAME_SIZE,
00040         sf::Color::Black,
00041         constants::normalGray,
00042         constants::hoverGray,
00043         constants::clickGray
00044     );
00045 }
00046 }
00047
00048 void MenuArray::resetActiveOptionsMenu() {
00049     this->activeOptionsMenu = constants::MenuArray::Button::NONE;
00050     this->activeCreateMode = constants::MenuArray::CreateMode::Button::NONE;
00051 }
00052
00053 void MenuArray::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
00054     if (this->activeOptionsMenu != constants::MenuArray::Button::NONE)
00055         this->buttons[this->activeOptionsMenu]->setColor(constants::normalGray);
00056
00057     for (int i = 0; i < constants::MenuArray::BUTTON_COUNT; ++i) {
00058         if (this->buttons[i]->pollEvent(mousePosView)) {
00059             std::cout << "Button " << i << " is clicked" << std::endl;
00060             this->activeOptionsMenu = static_cast<constants::MenuArray::Button>(i);
00061             this->activeAddMode = constants::MenuArray::AddMode::Textbox::NONE;
00062         }
00063     }
00064
00065     switch (this->activeOptionsMenu) {
00066     case constants::MenuArray::Button::CREATE_BUTTON:
00067         this->pollEventCreateMode(event, mousePosView);
00068         break;
00069     case constants::MenuArray::Button::ADD_BUTTON:
00070         this->pollEventAddMode(event, mousePosView);
00071         break;
00072     case constants::MenuArray::Button::DELETE_BUTTON:
00073         this->pollEventDeleteMode(event, mousePosView);
00074         break;
00075     case constants::MenuArray::Button::UPDATE_BUTTON:
00076         this->pollEventUpdateMode(event, mousePosView);
00077         break;
00078     case constants::MenuArray::Button::SEARCH_BUTTON:
00079         this->pollEventSearchMode(event, mousePosView);
00080         break;
00081     case constants::MenuArray::Button::ALLOCATE_BUTTON:
00082         this->pollEventAllocateMode(event, mousePosView);
00083         break;
00084     case constants::MenuArray::Button::NONE:
00085         break;
00086     }
00087 }
00088
00089 void MenuArray::update() {
00090     if (this->activeOptionsMenu != constants::MenuArray::Button::NONE)
00091         this->buttons[this->activeOptionsMenu]->setColor(constants::clickGreen);
00092
00093     for (Button* button : this->buttons) {
00094         button->update();
00095     }
00096
00097     switch (this->activeOptionsMenu) {
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 }
00120
00121 void MenuArray::render() {
00122     for (Button* button : this->buttons) {
00123         button->render();
00124     }
00125
00126     switch (this->activeOptionsMenu) {
00127         case constants::MenuArray::Button::CREATE_BUTTON:
00128             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 }
00149
00150 Button *MenuArray::getButton(int index) {
00151     return this->buttons[index];
00152 }
00153
00154 constants::MenuArray::Button MenuArray::getActiveOptionsMenu() {
00155     return this->activeOptionsMenu;
00156 }
00157
00158 constants::MenuArray::CreateMode::Button MenuArray::getActiveCreateMode() {
00159     return this->activeCreateMode;
00160 }
00161
00162 void MenuArray::initCreateMode() {
00163     // init stuff for create mode
00164     this->activeCreateMode = constants::MenuArray::CreateMode::Button::NONE;
00165     for (int i = 0; i < constants::MenuArray::CreateMode::BUTTON_COUNT; i++) {
00166         sf::Vector2f position = sf::Vector2f(
00167             this->buttons[0]->getPosition().x + (constants::optionButtonSize.x +
00168             constants::distance2ModeButtons) * static_cast<float>(i + 1),
00169             this->buttons[0]->getPosition().y
00170         );
00171         this->subCreateMode[i] = new Button(
00172             this->window,
00173             position,
00174             constants::optionButtonSize,
00175             constants::MenuArray::CreateMode::BUTTON_NAMES[i],
00176             constants::MenuArray::CreateMode::BUTTON_NAMES[i],
00177             constants::MenuArray::CreateMode::NAME_SIZE,
00178             sf::Color::Black,
00179             constants::normalGray,
00180             constants::hoverGray,
00181             constants::clickGray
00182         );
00183         if (i < 2)
00184             this->createTextbox[i] = new CustomTextbox(
00185                 this->window,
00186                 sf::Vector2f(
00187                     this->subCreateMode[0]->getPosition().x,
00188                     this->subCreateMode[0]->getPosition().y + constants::optionButtonSize.y +
00189                     constants::distance2ModeButtons
00190                 ),
00191                 20,
00192                 constants::MenuArray::CreateMode::TEXTBOX_NAMES[i],
00193                 constants::MenuArray::CreateMode::TEXTBOX_LENGTH[i],
00194             );
00195         this->createModeValue[i] = "None";
00196     }
00197     this->isOpenFileDialog = false;
00198 }
00199
00200 void MenuArray::pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView) {
00201     if (this->activeCreateMode != constants::MenuArray::CreateMode::Button::NONE)
00202         this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);
00203
00204     for (int i = 0; i < constants::MenuArray::CreateMode::BUTTON_COUNT; i++) {
00205         if (this->subCreateMode[i]->pollEvent(mousePosView)) {
00206             this->activeCreateMode = static_cast<constants::MenuArray::CreateMode::Button>(i);
00207         }
00208     }
00209 }

```

```

00204         if (i == constants::MenuArray::CreateMode::Button::FILE_BUTTON)
00205             this->isOpenFileDialog = true;
00206         std::cout << "Button " << i << " is clicked" << std::endl;
00207     }
00208 }
00209
00210 // this->testTextbox->pollEvent(event);
00211 if (this->activeCreateMode < constants::MenuArray::CreateMode::TEXTBOX_COUNT)
00212     this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
00213 }
00214 void MenuArray::updateCreateMode() {
00215     if (this->activeCreateMode != constants::MenuArray::CreateMode::Button::NONE)
00216         this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);
00217
00218     for (Button* button : this->subCreateMode) {
00219         button->update();
00220     }
00221
00222 // this->testTextbox->update();
00223 if (this->activeCreateMode < constants::MenuArray::CreateMode::TEXTBOX_COUNT) {
00224     this->createTextbox[this->activeCreateMode]->update();
00225     std::string inputUser = this->createTextbox[this->activeCreateMode]->getTextString();
00226     if (inputUser != "None") {
00227         std::cout << inputUser << std::endl;
00228         this->createTextbox[this->activeCreateMode]->resetInput();
00229     }
00230     this->createModeValue[this->activeCreateMode] = inputUser;
00231 } else if (this->activeCreateMode == constants::MenuArray::CreateMode::FILE_BUTTON) {
00232     if (this->isOpenFileDialog) {
00233         auto f = pfd::open_file("Choose files to read", pfd::path::home(),
00234                                {"Text Files (.txt .text)", "*.txt *.text",
00235                                 "All Files", "*"});
00236
00237         // wait for the user to select a file unless the window will be not responsive
00238         while (!f.ready(100)) {
00239             sf::Event event{};
00240             this->window->pollEvent(event);
00241         }
00242
00243         if (!f.result().empty()) {
00244             std::ifstream file(f.result()[0]);
00245             std::string line;
00246             file >> line;
00247             this->createModeValue[this->activeCreateMode] = line;
00248         }
00249     }
00250     this->isOpenFileDialog = false;
00251 }
00252 }
00253 void MenuArray::renderCreateMode() {
00254     for (Button* button : this->subCreateMode) {
00255         button->render();
00256     }
00257
00258 // this->testTextbox->render();
00259 if (this->activeCreateMode < constants::MenuArray::CreateMode::TEXTBOX_COUNT)
00260     this->createTextbox[this->activeCreateMode]->render();
00261 }
00262
00263 void MenuArray::initAddMode() {
00264     //init stuff for add mode
00265     this->activeAddMode = constants::MenuArray::AddMode::Textbox::NONE;
00266     for (int i = 0; i < constants::MenuArray::AddMode::TEXTBOX_COUNT; i++) {
00267         sf::Vector2f position = sf::Vector2f(
00268             this->buttons[1]->getPosition().x + (constants::optionButtonSize.x +
00269             constants::distance2ModeButtons),
00270             this->buttons[1]->getPosition().y
00271         );
00272         this->addTextbox[i] = new CustomTextbox{
00273             this->window,
00274             position,
00275             20,
00276             constants::MenuArray::AddMode::TEXTBOX_NAMES[i],
00277             constants::MenuArray::AddMode::TEXTBOX_LENGTH[i],
00278         };
00279         this->addModeValue[i] = "None";
00280     }
00281
00282 void MenuArray::pollEventAddMode(sf::Event event, sf::Vector2f mousePosView) {
00283     if (this->activeAddMode == constants::MenuArray::AddMode::NONE)
00284         this->activeAddMode = constants::MenuArray::AddMode::POSITION_TEXTBOX;
00285     this->addTextbox[this->activeAddMode]->pollEvent(event, mousePosView);
00286 }
00287 void MenuArray::updateAddMode() {
00288     if (this->activeAddMode == constants::MenuArray::AddMode::NONE)
00289         this->activeAddMode = constants::MenuArray::AddMode::POSITION_TEXTBOX;

```



```

00290
00291     this->addTextbox[this->activeAddMode]->update();
00292
00293     std::string inputUser = this->addTextbox[this->activeAddMode]->getTextString();
00294     // check if input is number
00295     bool isValid = true;
00296     for (char i : inputUser)
00297         if (!std::isdigit(i))
00298             isValid = false;
00299     if (isValid && inputUser != "None") {
00300         this->addModeValue[this->activeAddMode] = inputUser;
00301         std::cout << inputUser << std::endl;
00302         this->addTextbox[this->activeAddMode]->resetInput();
00303         this->activeAddMode =
00304             static_cast<constants::MenuArray::AddMode::Textbox>(!this->activeAddMode);
00305     }
00306 void MenuArray::renderAddMode() {
00307     this->addTextbox[this->activeAddMode]->render();
00308 }
00309
00310 void MenuArray::initDeleteMode() {
00311     sf::Vector2f position = sf::Vector2f(
00312         this->buttons[2]->getPosition().x + (constants::optionButtonSize.x +
00313         constants::distance2ModeButtons),
00314         this->buttons[2]->getPosition().y
00315     );
00316     this->deleteTextbox = new CustomTextbox{
00317         this->window,
00318         position,
00319         20,
00320         constants::MenuArray::DeleteMode::TEXTBOX_NAME,
00321         constants::MenuArray::DeleteMode::TEXTBOX_LENGTH,
00322     };
00323     this->deleteModeValue = "None";
00324 }
00325 void MenuArray::pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView) {
00326     this->deleteTextbox->pollEvent(event, mousePosView);
00327 }
00328 void MenuArray::updateDeleteMode() {
00329     this->deleteTextbox->update();
00330
00331     std::string inputUser = this->deleteTextbox->getTextString();
00332     // check if input is number
00333     bool isValid = true;
00334     for (char i : inputUser)
00335         if (!std::isdigit(i))
00336             isValid = false;
00337     if (isValid && inputUser != "None") {
00338         this->deleteModeValue = inputUser;
00339         std::cout << inputUser << std::endl;
00340         this->deleteTextbox->resetInput();
00341     }
00342 }
00343 void MenuArray::renderDeleteMode() {
00344     this->deleteTextbox->render();
00345 }
00346
00347 void MenuArray::initUpdateMode() {
00348     // init stuff for update mode
00349     this->activeUpdateMode = constants::MenuArray::UpdateMode::Textbox::NONE;
00350     for (int i = 0; i < constants::MenuArray::UpdateMode::TEXTBOX_COUNT; i++) {
00351         sf::Vector2f position = sf::Vector2f(
00352             this->buttons[3]->getPosition().x + (constants::optionButtonSize.x +
00353             constants::distance2ModeButtons),
00354             this->buttons[3]->getPosition().y
00355         );
00356         this->updateTextbox[i] = new CustomTextbox{
00357             this->window,
00358             position,
00359             20,
00360             constants::MenuArray::UpdateMode::TEXTBOX_NAMES[i],
00361             constants::MenuArray::UpdateMode::TEXTBOX_LENGTH[i],
00362         };
00363         this->updateModeValue[i] = "None";
00364     }
00365 }
00366 void MenuArray::pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView) {
00367     if (this->activeUpdateMode == constants::MenuArray::UpdateMode::NONE)
00368         this->activeUpdateMode = constants::MenuArray::UpdateMode::POSITION_TEXTBOX;
00369     this->updateTextbox[this->activeUpdateMode]->pollEvent(event, mousePosView);
00370 }
00371 void MenuArray::updateUpdateMode() {
00372     if (this->activeUpdateMode == constants::MenuArray::UpdateMode::NONE)
00373         this->activeUpdateMode = constants::MenuArray::UpdateMode::POSITION_TEXTBOX;
00374 }

```

```

00374     this->updateTextbox[this->activeUpdateMode]->update();
00375
00376     std::string inputUser = this->updateTextbox[this->activeUpdateMode]->getTextString();
00377     // check if input is number
00378     bool isValid = true;
00379     for (char i : inputUser)
00380         if (!std::isdigit(i))
00381             isValid = false;
00382     if (isValid && inputUser != "None") {
00383         this->updateModeValue[this->activeUpdateMode] = inputUser;
00384         std::cout << inputUser << std::endl;
00385         this->updateTextbox[this->activeUpdateMode]->resetInput();
00386         this->activeUpdateMode =
00387             static_cast<constants::MenuArray::UpdateMode::Textbox>(!this->activeUpdateMode);
00388     }
00389 void MenuArray::renderUpdateMode() {
00390     this->updateTextbox[this->activeUpdateMode]->render();
00391 }
00392
00393 void MenuArray::initSearchMode() {
00394     sf::Vector2f position = sf::Vector2f(
00395         this->buttons[4]->getPosition().x + (constants::optionButtonSize.x +
00396         constants::distance2ModeButtons),
00397         this->buttons[4]->getPosition().y
00398     );
00399     this->searchTextbox = new CustomTextbox{
00400         this->window,
00401         position,
00402         20,
00403         constants::MenuArray::SearchMode::TEXTBOX_NAME,
00404         constants::MenuArray::SearchMode::TEXTBOX_LENGTH,
00405     };
00406     this->searchModeValue = "None";
00407 }
00408 void MenuArray::pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView) {
00409     this->searchTextbox->pollEvent(event, mousePosView);
00410 }
00411 void MenuArray::updateSearchMode() {
00412     this->searchTextbox->update();
00413
00414     std::string inputUser = this->searchTextbox->getTextString();
00415     // check if input is number
00416     bool isValid = true;
00417     for (char i : inputUser)
00418         if (!std::isdigit(i))
00419             isValid = false;
00420     if (isValid && inputUser != "None") {
00421         this->searchModeValue = inputUser;
00422         std::cout << inputUser << std::endl;
00423         this->searchTextbox->resetInput();
00424     }
00425 }
00426 void MenuArray::renderSearchMode() {
00427     this->searchTextbox->render();
00428 }
00429 void MenuArray::initAllocateMode() {
00430     sf::Vector2f position = sf::Vector2f(
00431         this->buttons[5]->getPosition().x + (constants::optionButtonSize.x +
00432         constants::distance2ModeButtons),
00433         this->buttons[5]->getPosition().y
00434     );
00435     this->allocateTextbox = new CustomTextbox{
00436         this->window,
00437         position,
00438         20,
00439         constants::MenuArray::AllocateMode::TEXTBOX_NAME,
00440         constants::MenuArray::AllocateMode::TEXTBOX_LENGTH,
00441     };
00442     this->allocateModeValue = "None";
00443 }
00444 void MenuArray::pollEventAllocateMode(sf::Event event, sf::Vector2f mousePosView) {
00445     this->allocateTextbox->pollEvent(event, mousePosView);
00446 }
00447 void MenuArray::updateAllocateMode() {
00448     this->allocateTextbox->update();
00449
00450     std::string inputUser = this->allocateTextbox->getTextString();
00451     // check if input is number
00452     bool isValid = true;
00453     for (char i : inputUser)
00454         if (!std::isdigit(i))
00455             isValid = false;
00456     if (isValid && inputUser != "None") {
00457         this->allocateModeValue = inputUser;
00458         std::cout << inputUser << std::endl;

```

```

00458         this->allocateTextbox->resetInput();
00459     }
00460 }
00461 void MenuArray::renderAllocateMode() {
00462     this->allocateTextbox->render();
00463 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 01/05/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_MENUARRAY_HPP
00006 #define VISUALGO_CS162_MENUARRAY_HPP
00007
00008 #include <fstream>
00009 #include "Constants.hpp"
00010 #include "stuff/button.hpp"
00011 #include "stuff/CustomTextbox.hpp"
00012 #include "core/FileDialog.h"
00013
00014 class MenuArray {
00015 private:
00016     sf::RenderWindow* window;
00017     Button* buttons[constants::MenuArray::BUTTON_COUNT];
00018     constants::MenuArray::Type typeArray;
00019
00020     constants::MenuArray::Button activeOptionsMenu;
00021
00022     // stuff for create mode
00023     Button* subCreateMode[constants::MenuArray::CreateMode::BUTTON_COUNT];
00024     CustomTextbox* createTextbox[constants::MenuArray::CreateMode::BUTTON_COUNT];
00025     constants::MenuArray::CreateMode::Button activeCreateMode;
00026     bool isOpenFileDialog = false;
00027
00028     void initCreateMode();
00029     void pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView);
00030     void updateCreateMode();
00031     void renderCreateMode();
00032
00033     // stuff for add mode
00034     CustomTextbox* addTextbox[constants::MenuArray::AddMode::TEXTBOX_COUNT];
00035     constants::MenuArray::AddMode::Textbox activeAddMode;
00036
00037     void initAddMode();
00038     void pollEventAddMode(sf::Event event, sf::Vector2f mousePosView);
00039     void updateAddMode();
00040     void renderAddMode();
00041
00042     // stuff for delete mode
00043     CustomTextbox* deleteTextbox;
00044
00045     void initDeleteMode();
00046     void pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView);

```

```

00047     void updateDeleteMode();
00048     void renderDeleteMode();
00049
00050     // stuff for update mode
00051     CustomTextbox* updateTextbox[constants::MenuArray::UpdateMode::TEXTBOX_COUNT];
00052     constants::MenuArray::UpdateMode::Textbox activeUpdateMode;
00053
00054     void initUpdateMode();
00055     void pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView);
00056     void updateUpdateMode();
00057     void renderUpdateMode();
00058
00059     // stuff for search mode
00060     CustomTextbox* searchTextbox;
00061
00062     void initSearchMode();
00063     void pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView);
00064     void updateSearchMode();
00065     void renderSearchMode();
00066
00067     // stuff for allocate mode
00068     CustomTextbox* allocateTextbox;
00069
00070     void initAllocateMode();
00071     void pollEventAllocateMode(sf::Event event, sf::Vector2f mousePosView);
00072     void updateAllocateMode();
00073     void renderAllocateMode();
00074
00075     void init();
00076     void initButtons();
00077
00078 public:
00079     // stuff public for create mode
00080     std::string createModeValue[constants::MenuArray::CreateMode::BUTTON_COUNT];
00081     constants::MenuArray::CreateMode::Button getActiveCreateMode();
00082
00083     // stuff public for add mode
00084     std::string addModeValue[constants::MenuArray::AddMode::TEXTBOX_COUNT];
00085
00086     // stuff public for delete mode
00087     std::string deleteModeValue;
00088
00089     // stuff public for update mode
00090     std::string updateModeValue[constants::MenuArray::UpdateMode::TEXTBOX_COUNT];
00091
00092     // stuff public for search mode
00093     std::string searchModeValue;
00094
00095     // stuff public for allocate mode
00096     std::string allocateModeValue;
00097
00098     explicit MenuArray(sf::RenderWindow* window, constants::MenuArray::Type _typeArray);
00099     ~MenuArray() = default;
00100
00101     void resetActiveOptionsMenu();
00102
00103     void pollEvents(sf::Event event, sf::Vector2f mousePosView);
00104     void update();
00105     void render();
00106
00107     Button* getButton(int index);
00108     constants::MenuArray::Button getActiveOptionsMenu();
00109 };
00110
00111 #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.](#)

```
00001 //
```

```

00002 // Created by dirii on 26/04/2023.
00003 //
00004
00005 #include "MenuDataStructure.hpp"
00006
00007 MenuDataStructure::MenuDataStructure(sf::RenderWindow *window) {
00008     this->window = window;
00009     this->init();
00010 }
00011
00012 void MenuDataStructure::init() {
00013     this->initButtons();
00014     this->initCreateMode();
00015     this->initPushMode();
00016
00017     this->activeOptionsMenu = constants::MenuDataStructure::Button::NONE;
00018 }
00019
00020 void MenuDataStructure::initButtons() {
00021     for (int i = 0; i < constants::MenuDataStructure::BUTTON_COUNT; i++) {
00022         sf::Vector2f position = sf::Vector2f(
00023             constants::sideButtonSize.x + constants::distance2ModeButtons,
00024             constants::submenuButtonPos.y + (constants::optionButtonSize.y +
constants::distance2ModeButtons / 10) * static_cast<float>(i)
00025         );
00026         this->buttons[i] = new Button(
00027             this->window,
00028             position,
00029             constants::optionButtonSize,
00030             constants::MenuDataStructure::BUTTON_NAMES[i],
00031             constants::MenuDataStructure::BUTTON_NAMES[i],
00032             constants::MenuDataStructure::BUTTON_NAME_SIZE,
00033             sf::Color::Black,
00034             constants::normalGray,
00035             constants::hoverGray,
00036             constants::clickGray
00037         );
00038     }
00039 }
00040
00041 void MenuDataStructure::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
00042     if (this->activeOptionsMenu != constants::MenuDataStructure::Button::NONE)
00043         this->buttons[this->activeOptionsMenu]->setColor(constants::normalGray);
00044
00045     for (int i = 0; i < constants::MenuDataStructure::BUTTON_COUNT; i++) {
00046         if (this->buttons[i]->pollEvent(mousePosView)) {
00047             std::cout << "Button " << i << " is clicked" << std::endl;
00048             this->activeOptionsMenu = static_cast<constants::MenuDataStructure::Button>(i);
00049         }
00050     }
00051
00052     if (this->activeOptionsMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
00053         this->pollEventCreateMode(event, mousePosView);
00054     } else if (this->activeOptionsMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
00055         this->pollEventPushMode(event, mousePosView);
00056     }
00057 }
00058
00059 void MenuDataStructure::update() {
00060     if (this->activeOptionsMenu < constants::MenuDataStructure::Button::POP_BUTTON)
00061         this->buttons[this->activeOptionsMenu]->setColor(constants::clickGreen);
00062
00063     for (Button* button : this->buttons) {
00064         button->update();
00065     }
00066
00067     if (this->activeOptionsMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
00068         this->updateCreateMode();
00069     } else if (this->activeOptionsMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
00070         this->updatePushMode();
00071     }
00072 }
00073
00074 void MenuDataStructure::render() {
00075     for (Button* button : this->buttons) {
00076         button->render();
00077     }
00078
00079     if (this->activeOptionsMenu == constants::MenuDataStructure::Button::CREATE_BUTTON) {
00080         this->renderCreateMode();
00081     } else if (this->activeOptionsMenu == constants::MenuDataStructure::Button::PUSH_BUTTON) {
00082         this->renderPushMode();
00083     }
00084 }
00085
00086 Button *MenuDataStructure::getButton(int index) {
00087     return this->buttons[index];

```

```

00088 }
00089
00090 constants::MenuDataStructure::Button MenuDataStructure::getActiveOptionsMenu() {
00091     return this->activeOptionsMenu;
00092 }
00093
00094 constants::MenuDataStructure::CreateMode::Button MenuDataStructure::getActiveCreateMode() {
00095     return this->activeCreateMode;
00096 }
00097
00098 void MenuDataStructure::initCreateMode() {
00099     // init stuff for create mode
00100     this->activeCreateMode = constants::MenuDataStructure::CreateMode::Button::NONE;
00101     for (int i = 0; i < constants::MenuDataStructure::CreateMode::BUTTON_COUNT; i++) {
00102         sf::Vector2f position = sf::Vector2f(
00103             this->buttons[0]->getPosition().x + (constants::optionButtonSize.x +
00104             constants::distance2ModeButtons) * static_cast<float>(i + 1),
00105             this->buttons[0]->getPosition().y
00106         );
00107         this->subCreateMode[i] = new Button(
00108             this->window,
00109             position,
00110             constants::optionButtonSize,
00111             constants::MenuDataStructure::CreateMode::BUTTON_NAMES[i],
00112             constants::MenuDataStructure::CreateMode::BUTTON_NAMES[i],
00113             constants::MenuDataStructure::CreateMode::NAME_SIZE,
00114             sf::Color::Black,
00115             constants::normalGray,
00116             constants::hoverGray,
00117             constants::clickGray
00118         );
00119         if (i < 2)
00120             this->createTextbox[i] = new CustomTextbox(
00121                 this->window,
00122                 sf::Vector2f(
00123                     this->subCreateMode[0]->getPosition().x,
00124                     this->subCreateMode[0]->getPosition().y + constants::optionButtonSize.y +
00125                     constants::distance2ModeButtons
00126                 ),
00127                 20,
00128                 constants::MenuDataStructure::CreateMode::TEXTBOX_NAMES[i],
00129                 constants::MenuDataStructure::CreateMode::TEXTBOX_LENGTH[i],
00130             );
00131         this->createModeValue[i] = "None";
00132     }
00133     this->isOpenFileDialog = false;
00134 }
00135
00136 void MenuDataStructure::pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView) {
00137     if (this->activeCreateMode != constants::MenuDataStructure::CreateMode::Button::NONE)
00138         this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);
00139     for (int i = 0; i < constants::MenuDataStructure::CreateMode::BUTTON_COUNT; i++) {
00140         if (this->subCreateMode[i]->pollEvent(mousePosView)) {
00141             this->activeCreateMode = static_cast<constants::MenuDataStructure::CreateMode::Button>(i);
00142             if (i == constants::MenuDataStructure::CreateMode::Button::FILE_BUTTON)
00143                 this->isOpenFileDialog = true;
00144             std::cout << "Button " << i << " is clicked" << std::endl;
00145         }
00146     }
00147     if (this->activeCreateMode < constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT)
00148         this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
00149 }
00150
00151 void MenuDataStructure::updateCreateMode() {
00152     if (this->activeCreateMode != constants::MenuDataStructure::CreateMode::Button::NONE)
00153         this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);
00154     for (Button* button : this->subCreateMode) {
00155         button->update();
00156     }
00157 }
00158
00159 // this->testTextbox->update();
00160 if (this->activeCreateMode < constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT) {
00161     this->createTextbox[this->activeCreateMode]->update();
00162     std::string inputUser = this->createTextbox[this->activeCreateMode]->getTextString();
00163     if (inputUser != "None") {
00164         std::cout << inputUser << std::endl;
00165         this->createTextbox[this->activeCreateMode]->resetInput();
00166     }
00167     this->createModeValue[this->activeCreateMode] = inputUser;
00168 } else if (this->activeCreateMode == constants::MenuDataStructure::CreateMode::FILE_BUTTON) {
00169     if (this->isOpenFileDialog) {
00170         auto f = pfd::open_file("Choose files to read", pfd::path::home(),
00171             {"Text Files (.txt .text)", "*.txt *.text",
00172             "All Files", "*"});
00172     }

```

```

00173
00174         // wait for the user to select a file unless the window will be not responsive
00175         while (!f.ready(100)) {
00176             sf::Event event{};
00177             this->window->pollEvent(event);
00178         }
00179
00180         if (!f.result().empty()) {
00181             std::ifstream file(f.result()[0]);
00182             std::string line;
00183             file >> line;
00184             this->createModeValue[this->activeCreateMode] = line;
00185         }
00186     }
00187     this->isOpenFileDialog = false;
00188 }
00189 }
00190
00191 void MenuDataStructure::renderCreateMode() {
00192     for (Button* button : this->subCreateMode) {
00193         button->render();
00194     }
00195
00196     if (this->activeCreateMode < constants::MenuDataStructure::CreateMode::TEXTBOX_COUNT)
00197         this->createTextbox[this->activeCreateMode]->render();
00198 }
00199
00200 void MenuDataStructure::initPushMode() {
00201     sf::Vector2f position = sf::Vector2f(
00202         this->buttons[1]->getPosition().x + (constants::optionButtonSize.x +
00203         constants::distance2ModeButtons),
00204         this->buttons[1]->getPosition().y
00205     );
00206     this->pushTextbox = new CustomTextbox{
00207         this->window,
00208         position,
00209         20,
00210         constants::MenuDataStructure::PushMode::TEXTBOX_NAME,
00211         constants::MenuDataStructure::PushMode::TEXTBOX_LENGTH,
00212     };
00213     this->pushModeValue = "None";
00214 }
00215 void MenuDataStructure::pollEventPushMode(sf::Event event, sf::Vector2f mousePosView) {
00216     this->pushTextbox->pollEvent(event, mousePosView);
00217 }
00218
00219 void MenuDataStructure::updatePushMode() {
00220     this->pushTextbox->update();
00221
00222     std::string inputUser = this->pushTextbox->getTextString();
00223     // check if input is number
00224     bool isValid = true;
00225     for (char i : inputUser)
00226         if (!std::isdigit(i))
00227             isValid = false;
00228     if (isValid && inputUser != "None") {
00229         this->pushModeValue = inputUser;
00230         std::cout << inputUser << std::endl;
00231         this->pushTextbox->resetInput();
00232     }
00233 }
00234
00235 void MenuDataStructure::renderPushMode() {
00236     this->pushTextbox->render();
00237 }
00238
00239 void MenuDataStructure::resetActiveOptionsMenu() {
00240     this->activeOptionsMenu = constants::MenuDataStructure::Button::NONE;
00241     this->activeCreateMode = constants::MenuDataStructure::CreateMode::Button::NONE;
00242 }
00243
00244 void MenuDataStructure::resetActiveOptionsMenuOnly() {
00245     this->activeOptionsMenu = constants::MenuDataStructure::Button::NONE;
00246 }

```

## 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.](#)

```
00001 //
00002 // Created by dirii on 26/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_MENUDATASTRUCTURE_HPP
00006 #define VISUALGO_CS162_MENUDATASTRUCTURE_HPP
00007
00008 #include <fstream>
00009 #include "Constants.hpp"
00010 #include "stuff/button.hpp"
00011 #include "stuff/CustomTextbox.hpp"
00012 #include "core/FileDialog.h"
00013
00014 class MenuDataStructure {
00015 private:
00016     sf::RenderWindow* window;
00017     Button* buttons[constants::MenuDataStructure::BUTTON_COUNT];
00018
00019     constants::MenuDataStructure::Button activeOptionsMenu;
00020
00021     // stuff for create mode
00022     Button* subCreateMode[constants::MenuDataStructure::CreateMode::BUTTON_COUNT];
00023     CustomTextbox* createTextbox[constants::MenuDataStructure::CreateMode::BUTTON_COUNT];
00024     constants::MenuDataStructure::CreateMode::Button activeCreateMode;
00025     bool isOpenFileDialog = false;
00026
00027     void initCreateMode();
00028     void pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView);
00029     void updateCreateMode();
00030     void renderCreateMode();
00031
00032     // stuff for push mode
00033     CustomTextbox* pushTextbox;
00034
00035     void initPushMode();
00036     void pollEventPushMode(sf::Event event, sf::Vector2f mousePosView);
00037     void updatePushMode();
00038     void renderPushMode();
00039
00040     void init();
00041     void initButtons();
00042
00043 public:
00044     // stuff public for create mode
00045     std::string createModeValue[constants::MenuDataStructure::CreateMode::BUTTON_COUNT];
00046     constants::MenuDataStructure::CreateMode::Button getActiveCreateMode();
00047
00048     // stuff public for push mode
00049     std::string pushModeValue;
00050
00051     explicit MenuDataStructure(sf::RenderWindow* window);
00052     ~MenuDataStructure() = default;
00053
00054     void resetActiveOptionsMenu();
00055     void resetActiveOptionsMenuOnly();
00056
00057     void pollEvents(sf::Event event, sf::Vector2f mousePosView);
00058     void update();
00059     void render();
00060
00061     Button* getButton(int index);
00062     constants::MenuDataStructure::Button getActiveOptionsMenu();
00063 };
00064
00065 #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.](#)

```
00001 //
00002 // Created by dirii on 30/03/2023.
00003 //
00004
00005 #include "MenuLinkedList.hpp"
00006
00007 void MenuLinkedList::init() {
00008     this->initButtons();
00009     this->initCreateMode();
00010     this->initAddMode();
00011     this->initDeleteMode();
00012     this->initUpdateMode();
00013     this->initSearchMode();
00014
00015     this->activeOptionsMenu = constants::MenuLinkedList::Button::NONE;
00016 }
00017
00018 void MenuLinkedList::initButtons() {
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 +
00023                 constants::distance2ModeButtons / 10) * static_cast<float>(i)
00024         );
00025         this->buttons[i] = new Button(
00026             this->window,
00027             position,
00028             constants::optionButtonSize,
00029             constants::MenuLinkedList::BUTTON_NAMES[i],
00030             constants::MenuLinkedList::BUTTON_NAMES[i],
00031             constants::MenuLinkedList::BUTTON_NAME_SIZE,
00032             sf::Color::Black,
00033             constants::normalGray,
00034             constants::hoverGray,
00035             constants::clickGray
00036         );
00037     }
00038 }
00039 MenuLinkedList::MenuLinkedList(sf::RenderWindow *window) {
00040     this->window = window;
00041     this->init();
00042 }
00043
00044 void MenuLinkedList::pollEvents(sf::Event event, sf::Vector2f mousePosView) {
00045     if (this->activeOptionsMenu != constants::MenuLinkedList::Button::NONE)
00046         this->buttons[this->activeOptionsMenu]->setColor(constants::normalGray);
00047
00048     for (int i = 0; i < constants::MenuLinkedList::BUTTON_COUNT; i++) {
00049         if (this->buttons[i]->pollEvent(mousePosView)) {
00050             std::cout << "Button " << i << " is clicked" << std::endl;
00051             this->activeOptionsMenu = static_cast<constants::MenuLinkedList::Button>(i);
00052             this->activeAddMode = constants::MenuLinkedList::AddMode::Textbox::NONE;
00053         }
00054     }
00055
00056     switch (this->activeOptionsMenu) {
00057         case constants::MenuLinkedList::Button::CREATE_BUTTON:
00058             this->pollEventCreateMode(event, mousePosView);
00059             break;
00060         case constants::MenuLinkedList::Button::ADD_BUTTON:
00061             this->pollEventAddMode(event, mousePosView);
00062             break;
00063         case constants::MenuLinkedList::Button::DELETE_BUTTON:
00064             this->pollEventDeleteMode(event, mousePosView);
00065             break;
00066         case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00067             this->pollEventUpdateMode(event, mousePosView);
00068             break;
00069         case constants::MenuLinkedList::Button::SEARCH_BUTTON:
```

```

00070         this->pollEventSearchMode(event, mousePosView);
00071         break;
00072     case constants::MenuLinkedList::Button::NONE:
00073         break;
00074     }
00075 }
00076
00077 void MenuLinkedList::update() {
00078     if (this->activeOptionsMenu != constants::MenuLinkedList::Button::NONE)
00079         this->buttons[this->activeOptionsMenu]->setColor(constants::clickGreen);
00080
00081     for (Button* button : this->buttons) {
00082         button->update();
00083     }
00084
00085     switch (this->activeOptionsMenu) {
00086     case constants::MenuLinkedList::Button::CREATE_BUTTON:
00087         this->updateCreateMode();
00088         break;
00089     case constants::MenuLinkedList::Button::ADD_BUTTON:
00090         this->updateAddMode();
00091         break;
00092     case constants::MenuLinkedList::Button::DELETE_BUTTON:
00093         this->updateDeleteMode();
00094         break;
00095     case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00096         this->updateUpdateMode();
00097         break;
00098     case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00099         this->updateSearchMode();
00100         break;
00101     case constants::MenuLinkedList::Button::NONE:
00102         break;
00103     }
00104 }
00105
00106 void MenuLinkedList::render() {
00107     for (Button* button : this->buttons) {
00108         button->render();
00109     }
00110
00111     switch (this->activeOptionsMenu) {
00112     case constants::MenuLinkedList::Button::CREATE_BUTTON:
00113         this->renderCreateMode();
00114         break;
00115     case constants::MenuLinkedList::Button::ADD_BUTTON:
00116         this->renderAddMode();
00117         break;
00118     case constants::MenuLinkedList::Button::DELETE_BUTTON:
00119         this->renderDeleteMode();
00120         break;
00121     case constants::MenuLinkedList::Button::UPDATE_BUTTON:
00122         this->renderUpdateMode();
00123         break;
00124     case constants::MenuLinkedList::Button::SEARCH_BUTTON:
00125         this->renderSearchMode();
00126         break;
00127     case constants::MenuLinkedList::Button::NONE:
00128         break;
00129     }
00130 }
00131
00132 Button *MenuLinkedList::getButton(int index) {
00133     return this->buttons[index];
00134 }
00135
00136 void MenuLinkedList::resetActiveOptionsMenu() {
00137     this->activeOptionsMenu = constants::MenuLinkedList::Button::NONE;
00138     this->activeCreateMode = constants::MenuLinkedList::CreateMode::Button::NONE;
00139 }
00140
00141 void MenuLinkedList::initCreateMode() {
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 }
00176 void MenuLinkedList::pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView) {
00177     if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
00178         this->subCreateMode[this->activeCreateMode]->setColor(constants::normalGray);
00179
00180     for (int i = 0; i < constants::MenuLinkedList::CreateMode::BUTTON_COUNT; i++) {
00181         if (this->subCreateMode[i]->pollEvent(mousePosView)) {
00182             this->activeCreateMode = static_cast<constants::MenuLinkedList::CreateMode::Button>(i);
00183             if (i == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON)
00184                 this->isOpenFileDialog = true;
00185             std::cout << "Button " << i << " is clicked" << std::endl;
00186         }
00187     }
00188
00189     // this->testTextbox->pollEvent(event);
00190     if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
00191         this->createTextbox[this->activeCreateMode]->pollEvent(event, mousePosView);
00192 }
00193 void MenuLinkedList::updateCreateMode() {
00194     if (this->activeCreateMode != constants::MenuLinkedList::CreateMode::Button::NONE)
00195         this->subCreateMode[this->activeCreateMode]->setColor(constants::clickGreen);
00196
00197     for (Button* button : this->subCreateMode) {
00198         button->update();
00199     }
00200
00201     // this->testTextbox->update();
00202     if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT) {
00203         this->createTextbox[this->activeCreateMode]->update();
00204         std::string userInput = this->createTextbox[this->activeCreateMode]->getTextString();
00205         if (inputUser != "None") {
00206             std::cout << inputUser << std::endl;
00207             this->createTextbox[this->activeCreateMode]->resetInput();
00208         }
00209         this->createModeValue[this->activeCreateMode] = userInput;
00210     } else if (this->activeCreateMode == constants::MenuLinkedList::CreateMode::FILE_BUTTON) {
00211         if (this->isOpenFileDialog) {
00212             auto f = pfd::open_file("Choose files to read", pfd::path::home(),
00213                 {"Text Files (.txt .text)", "*.txt *.text",
00214                 "All Files", "*"});
00215
00216             // wait for the user to select a file unless the window will be not responsive
00217             while (!f.ready(100)) {
00218                 sf::Event event{};
00219                 this->window->pollEvent(event);
00220             }
00221
00222             if (!f.result().empty()) {
00223                 std::ifstream file(f.result()[0]);
00224                 std::string line;
00225                 file >> line;
00226                 this->createModeValue[this->activeCreateMode] = line;
00227             }
00228         }
00229         this->isOpenFileDialog = false;
00230     }
00231 }
00232 void MenuLinkedList::renderCreateMode() {
00233     for (Button* button : this->subCreateMode) {
00234         button->render();
00235     }
00236
00237     // this->testTextbox->render();
00238     if (this->activeCreateMode < constants::MenuLinkedList::CreateMode::TEXTBOX_COUNT)
00239         this->createTextbox[this->activeCreateMode]->render();
00240 }
00241

```

```

00242 constants::MenuLinkedList::CreateMode::Button MenuLinkedList::getActiveCreateMode() {
00243     return this->activeCreateMode;
00244 }
00245
00246 constants::MenuLinkedList::Button MenuLinkedList::getActiveOptionsMenu() {
00247     return this->activeOptionsMenu;
00248 }
00249
00250 void MenuLinkedList::initAddMode() {
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 }
00268 void MenuLinkedList::pollEventAddMode(sf::Event event, sf::Vector2f mousePosView) {
00269     if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
00270         this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;
00271
00272     this->addTextbox[this->activeAddMode]->pollEvent(event, mousePosView);
00273 }
00274 void MenuLinkedList::updateAddMode() {
00275     if (this->activeAddMode == constants::MenuLinkedList::AddMode::NONE)
00276         this->activeAddMode = constants::MenuLinkedList::AddMode::POSITION_TEXTBOX;
00277
00278     this->addTextbox[this->activeAddMode]->update();
00279
00280     std::string inputUser = this->addTextbox[this->activeAddMode]->getTextString();
00281     // check if input is number
00282     bool isValid = true;
00283     for (char i : inputUser)
00284         if (!std::isdigit(i))
00285             isValid = false;
00286     if (isValid && inputUser != "None") {
00287         this->addModeValue[this->activeAddMode] = inputUser;
00288         std::cout << inputUser << std::endl;
00289         this->addTextbox[this->activeAddMode]->resetInput();
00290         this->activeAddMode =
static_cast<constants::MenuLinkedList::AddMode::Textbox>(!this->activeAddMode);
00291     }
00292 }
00293 void MenuLinkedList::renderAddMode() {
00294     this->addTextbox[this->activeAddMode]->render();
00295 }
00296
00297 void MenuLinkedList::initDeleteMode() {
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 }
00311 void MenuLinkedList::pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView) {
00312     this->deleteTextbox->pollEvent(event, mousePosView);
00313 }
00314 void MenuLinkedList::updateDeleteMode() {
00315     this->deleteTextbox->update();
00316
00317     std::string inputUser = this->deleteTextbox->getTextString();
00318     // check if input is number
00319     bool isValid = true;
00320     for (char i : inputUser)
00321         if (!std::isdigit(i))
00322             isValid = false;
00323     if (isValid && inputUser != "None") {
00324         this->deleteModeValue = inputUser;
00325         std::cout << inputUser << std::endl;

```

```

00326         this->deleteTextbox->resetInput();
00327     }
00328 }
00329 void MenuLinkedList::renderDeleteMode() {
00330     this->deleteTextbox->render();
00331 }
00332
00333 void MenuLinkedList::initUpdateMode() {
00334     // init stuff for update mode
00335     this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::Textbox::NONE;
00336     for (int i = 0; i < constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT; i++) {
00337         sf::Vector2f position = sf::Vector2f(
00338             this->buttons[3]->getPosition().x + (constants::optionButtonSize.x +
00339             constants::distance2ModeButtons),
00340             this->buttons[3]->getPosition().y
00341         );
00342         this->updateTextbox[i] = new CustomTextbox{
00343             this->window,
00344             position,
00345             20,
00346             constants::MenuLinkedList::UpdateMode::TEXTBOX_NAMES[i],
00347             constants::MenuLinkedList::UpdateMode::TEXTBOX_LENGTH[i],
00348         };
00349         this->updateModeValue[i] = "None";
00350     }
00351 }
00352 void MenuLinkedList::pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView) {
00353     if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
00354         this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;
00355     this->updateTextbox[this->activeUpdateMode]->pollEvent(event, mousePosView);
00356 }
00357 void MenuLinkedList::updateUpdateMode() {
00358     if (this->activeUpdateMode == constants::MenuLinkedList::UpdateMode::NONE)
00359         this->activeUpdateMode = constants::MenuLinkedList::UpdateMode::POSITION_TEXTBOX;
00360     this->updateTextbox[this->activeUpdateMode]->update();
00361
00362     std::string inputUser = this->updateTextbox[this->activeUpdateMode]->getTextString();
00363     // check if input is number
00364     bool isValid = true;
00365     for (char i : inputUser)
00366         if (!std::isdigit(i))
00367             isValid = false;
00368     if (isValid && inputUser != "None") {
00369         this->updateModeValue[this->activeUpdateMode] = inputUser;
00370         std::cout << inputUser << std::endl;
00371         this->updateTextbox[this->activeUpdateMode]->resetInput();
00372         this->activeUpdateMode =
00373             static_cast<constants::MenuLinkedList::UpdateMode::Textbox>(!this->activeUpdateMode);
00374     }
00375 }
00376 void MenuLinkedList::renderUpdateMode() {
00377     this->updateTextbox[this->activeUpdateMode]->render();
00378 }
00379
00380 void MenuLinkedList::initSearchMode() {
00381     sf::Vector2f position = sf::Vector2f(
00382         this->buttons[4]->getPosition().x + (constants::optionButtonSize.x +
00383         constants::distance2ModeButtons),
00384         this->buttons[4]->getPosition().y
00385     );
00386     this->searchTextbox = new CustomTextbox{
00387         this->window,
00388         position,
00389         20,
00390         constants::MenuLinkedList::SearchMode::TEXTBOX_NAME,
00391         constants::MenuLinkedList::SearchMode::TEXTBOX_LENGTH,
00392     };
00393     this->searchModeValue = "None";
00394 }
00395 void MenuLinkedList::pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView) {
00396     this->searchTextbox->pollEvent(event, mousePosView);
00397 }
00398 void MenuLinkedList::updateSearchMode() {
00399     this->searchTextbox->update();
00400
00401     std::string inputUser = this->searchTextbox->getTextString();
00402     // check if input is number
00403     bool isValid = true;
00404     for (char i : inputUser)
00405         if (!std::isdigit(i))
00406             isValid = false;
00407     if (isValid && inputUser != "None") {
00408         this->searchModeValue = inputUser;
00409         std::cout << inputUser << std::endl;
00410         this->searchTextbox->resetInput();
00411     }

```

```

00410     }
00411 }
00412 void MenuLinkedList::renderSearchMode() {
00413     this->searchTextbox->render();
00414 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 30/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_MENULINKEDLIST_HPP
00006 #define VISUALGO_CS162_MENULINKEDLIST_HPP
00007
00008 #include <iostream>
00009 #include <fstream>
00010 #include <SFML/Graphics.hpp>
00011 #include "core/FileDialog.h"
00012 #include "stuff/button.hpp"
00013 #include "stuff/CustomTextbox.hpp"
00014 #include "Constants.hpp"
00015
00016 class MenuLinkedList {
00017 protected:
00018     sf::RenderWindow* window;
00019     Button* buttons[constants::MenuLinkedList::BUTTON_COUNT];
00020
00021     constants::MenuLinkedList::Button activeOptionsMenu;
00022
00023     // stuff for create mode
00024     Button* subCreateMode[constants::MenuLinkedList::CreateMode::BUTTON_COUNT];
00025     CustomTextbox* createTextbox[constants::MenuLinkedList::CreateMode::BUTTON_COUNT];
00026     constants::MenuLinkedList::CreateMode::Button activeCreateMode;
00027     bool isOpenFileDialog = false;
00028
00029     void initCreateMode();
00030     void pollEventCreateMode(sf::Event event, sf::Vector2f mousePosView);
00031     void updateCreateMode();
00032     void renderCreateMode();
00033
00034     // stuff for add mode
00035     CustomTextbox* addTextbox[constants::MenuLinkedList::AddMode::TEXTBOX_COUNT];
00036     constants::MenuLinkedList::AddMode::Textbox activeAddMode;
00037
00038     void initAddMode();
00039     void pollEventAddMode(sf::Event event, sf::Vector2f mousePosView);
00040     void updateAddMode();
00041     void renderAddMode();
00042
00043     // stuff for delete mode
00044     CustomTextbox* deleteTextbox;

```

```

00045
00046     void initDeleteMode();
00047     void pollEventDeleteMode(sf::Event event, sf::Vector2f mousePosView);
00048     void updateDeleteMode();
00049     void renderDeleteMode();
00050
00051     // stuff for update mode
00052     CustomTextbox* updateTextbox[constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT];
00053     constants::MenuLinkedList::UpdateMode::Textbox activeUpdateMode;
00054
00055     void initUpdateMode();
00056     void pollEventUpdateMode(sf::Event event, sf::Vector2f mousePosView);
00057     void updateUpdateMode();
00058     void renderUpdateMode();
00059
00060     // stuff for search mode
00061     CustomTextbox* searchTextbox;
00062
00063     void initSearchMode();
00064     void pollEventSearchMode(sf::Event event, sf::Vector2f mousePosView);
00065     void updateSearchMode();
00066     void renderSearchMode();
00067
00068     void init();
00069     void initButtons();
00070
00071 public:
00072     // stuff public for create mode
00073     std::string createModeValue[constants::MenuLinkedList::CreateMode::BUTTON_COUNT];
00074     constants::MenuLinkedList::CreateMode::Button getActiveCreateMode();
00075
00076     // stuff public for add mode
00077     std::string addModeValue[constants::MenuLinkedList::AddMode::TEXTBOX_COUNT];
00078
00079     // stuff public for delete mode
00080     std::string deleteModeValue;
00081
00082     // stuff public for update mode
00083     std::string updateModeValue[constants::MenuLinkedList::UpdateMode::TEXTBOX_COUNT];
00084
00085     // stuff public for search mode
00086     std::string searchModeValue;
00087
00088     explicit MenuLinkedList(sf::RenderWindow* window);
00089     ~MenuLinkedList() = default;
00090
00091     void resetActiveOptionsMenu();
00092
00093     void pollEvents(sf::Event event, sf::Vector2f mousePosView);
00094     void update();
00095     void render();
00096
00097     Button* getButton(int index);
00098     constants::MenuLinkedList::Button getActiveOptionsMenu();
00099 };
00100
00101 #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.](#)

```

00001 //
00002 // Created by dirii on 29/03/2023.
00003 //
00004
00005 #include "QueueScene.hpp"
00006
00007 QueueScene::QueueScene(sf::RenderWindow *window) : BaseScene(window) {
00008     this->init();
00009 }

```

```

00010
00011 void QueueScene::update() {
00012     if (this->isMenuOpen) {
00013         this->menu->update();
00014
00015         constants::MenuDataStructure::Button status = this->menu->getActiveOptionsMenu();
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                     this->linkedList->getSize(),
00058                     this->menu->pushModeValue,
00059                     this->pushModeEvents(this->linkedList->getSize())
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->getActiveOptionsMenu() !=
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->resetActiveOptionsMenuOnly();
00077                 this->controlMenu->reset();
00078                 break;
00079             case constants::MenuDataStructure::Button::CLEAR_BUTTON:
00080                 if (this->menu->getActiveOptionsMenu() !=
constants::MenuDataStructure::Button::CLEAR_BUTTON)
00081                     break;
00082
00083                 this->linkedList->createLinkedList(0);
00084
00085                 std::cout << "Cleared " << std::endl;
00086                 this->menu->resetActiveOptionsMenuOnly();
00087                 this->controlMenu->reset();
00088                 break;
00089         }
00090     }
00091
00092     this->controlMenu->update();

```



```

00093
00094     this->linkedList->processControlMenu(this->controlMenu->getStatus());
00095     this->linkedList->setSpeed(this->controlMenu->getSpeed());
00096
00097     this->linkedList->update();
00098 }
00099
00100 void QueueScene::render() {
00101     if (this->isMenuOpen)
00102         this->menu->render();
00103
00104     if (this->isDemoCodeOpen)
00105         this->linkedList->renderHighlighter();
00106
00107     this->controlMenu->render();
00108     this->linkedList->render();
00109 }
00110
00111 void QueueScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00112     if (this->isMenuOpen)
00113         this->menu->pollEvents(event, mousePosView);
00114
00115     this->controlMenu->pollEvents(event, mousePosView);
00116 }
00117
00118 void QueueScene::init() {
00119     this->menu = new MenuDataStructure(this->window);
00120     this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::SINGLY);
00121 }
00122
00123 void QueueScene::reset() {
00124     this->menu->resetActiveOptionsMenu();
00125 }
00126
00127 std::vector<EventAnimation> QueueScene::pushModeEvents(int chosenNode) {
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"},
00200                 {i, "current"}
00201             };
00202             event.colorNodes.push_back(i);
00203             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00204             if (chosenNode == this->linkedList->getSize())
00205                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00206             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00207             event.lines = {6};
00208
00209             events.emplace_back(event);
00210
00211             if (i == chosenNode - 1) break;
00212
00213             event.reset();
00214             event.titleNodes = {
00215                 {0, "head"},
00216                 {chosenNode, "temp"},
00217                 {i, "current"}
00218             };
00219             event.colorNodes.push_back(i);
00220             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00221             event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00222             if (chosenNode == this->linkedList->getSize())
00223                 event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00224             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00225             event.lines = {7};
00226
00227             events.emplace_back(event);
00228         }
00229
00230         if (chosenNode != this->linkedList->getSize()) {
00231             event.reset();
00232             event.titleNodes = {
00233                 {0, "head"},
00234                 {chosenNode, "temp"},
00235                 {chosenNode - 1, "current"}
00236             };
00237             event.colorNodes.push_back(chosenNode);
00238             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00239             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00240             event.isPrintNormal = true;
00241             event.lines = {8};
00242
00243             events.emplace_back(event);
00244         }
00245
00246         event.reset();
00247         event.titleNodes = {
00248             {0, "head"},
00249             {chosenNode, "temp"}
00250         };
00251         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00252         event.lines = {9};
00253
00254         events.emplace_back(event);
00255     }
00256
00257     return events;
00258 }
00259
00260 std::vector<EventAnimation> QueueScene::popModeEvents(int chosenNode) {
00261     this->linkedList->resetEvents();
00262     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00263         return {};
00264
00265     this->linkedList->initHighlighter(
00266         constants::Highlighter::SLL::CODES_PATH[1].second,

```

```

00267         constants::Highlighter::SLL::CODES_PATH[1].first
00268     );
00269
00270     std::vector<EventAnimation> events;
00271     EventAnimation event;
00272
00273     if (!chosenNode) {
00274         event.titleNodes.emplace_back(chosenNode, "head|temp");
00275         event.colorNodes.push_back(chosenNode);
00276         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00277         event.lines = {0, 1};
00278
00279         events.emplace_back(event);
00280
00281         if (this->linkedList->getSize() > 1) {
00282             event.reset();
00283             event.titleNodes = {
00284                 {chosenNode, "temp"},
00285                 {1, "head"}
00286             };
00287             event.colorNodes.push_back(1);
00288             event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00289             event.isPrintNormal = true;
00290             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00291             event.lines = {2};
00292
00293             events.emplace_back(event);
00294         }
00295
00296         event.reset();
00297         event.titleNodes.emplace_back(1, "head");
00298         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00299         event.lines = {3};
00300
00301         events.emplace_back(event);
00302     } else {
00303         event.reset();
00304         event.titleNodes.emplace_back(0, "head|current");
00305         event.colorNodes.push_back(0);
00306         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00307         event.lines = {5};
00308
00309         events.emplace_back(event);
00310
00311         for (int i = 0; i < chosenNode; ++i) {
00312             event.reset();
00313             event.titleNodes = {
00314                 {0, "head"},
00315                 {i, "current"}
00316             };
00317             event.colorNodes.push_back(i);
00318             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00319             event.lines = {6};
00320
00321             events.emplace_back(event);
00322
00323             if (i == chosenNode - 1) break;
00324
00325             event.reset();
00326             event.titleNodes = {
00327                 {0, "head"},
00328                 {i, "current"}
00329             };
00330             event.colorNodes.push_back(i);
00331             event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00332             event.statusChosenNode = NodeInfo::StatusNode::InChain;
00333             event.lines = {7};
00334
00335             events.emplace_back(event);
00336         }
00337
00338         event.reset();
00339         event.titleNodes = {
00340             {0, "head"},
00341             {chosenNode, "temp"},
00342             {chosenNode - 1, "current"}
00343         };
00344         event.colorNodes.push_back(chosenNode);
00345         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00346         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00347         event.lines = {8};
00348
00349         events.emplace_back(event);
00350
00351         if (chosenNode != this->linkedList->getSize() - 1) {
00352             event.reset();
00353             event.titleNodes = {

```

```

00354         {0, "head"},
00355         {chosenNode, "temp"},
00356         {chosenNode - 1, "current"}
00357     };
00358     event.colorNodes.push_back(chosenNode);
00359     event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00360     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00361     event.isPrintNormal = true;
00362     event.lines = {9};
00363
00364     events.emplace_back(event);
00365
00366     event.reset();
00367     event.titleNodes.emplace_back(0, "head");
00368     event.statusChosenNode = NodeInfo::StatusNode::Visible;
00369     event.lines = {10};
00370
00371     events.emplace_back(event);
00372 } else {
00373     event.reset();
00374     event.titleNodes = {
00375         {0, "head"},
00376         {chosenNode, "temp"},
00377         {chosenNode - 1, "current"}
00378     };
00379     event.colorNodes.push_back(chosenNode);
00380     event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00381     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00382     event.lines = {9};
00383
00384     events.emplace_back(event);
00385
00386     event.reset();
00387     event.titleNodes.emplace_back(0, "head");
00388     event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00389     event.statusChosenNode = NodeInfo::StatusNode::Visible;
00390     event.lines = {10};
00391
00392     events.emplace_back(event);
00393 }
00394 }
00395
00396 return events;
00397 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 29/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_QUEUESCENE_HPP
00006 #define VISUALGO_CS162_QUEUESCENE_HPP
00007
00008 #include "BaseScene.hpp"
00009 #include "MenuDataStructure.hpp"
00010 #include "core/LinkedList.hpp"
00011

```

```

00012 class QueueScene : public BaseScene{
00013 private:
00014     MenuDataStructure* menu;
00015     LinkedList* linkedList;
00016
00017     void init();
00018
00019 public:
00020     explicit QueueScene(sf::RenderWindow* window);
00021
00022     void reset();
00023
00024     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00025     void update() override;
00026     void render() override;
00027
00028     std::vector<EventAnimation> pushModeEvents(int chosenNode);
00029     std::vector<EventAnimation> popModeEvents(int chosenNode);
00030 };
00031
00032 #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.](#)

```

00001 //
00002 // Created by dirii on 26/03/2023.
00003 //
00004
00005 #include "SLLScene.hpp"
00006
00007 void SLLScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00008     if (this->isMenuOpen)
00009         this->menu->pollEvents(event, mousePosView);
00010
00011     this->controlMenu->pollEvents(event, mousePosView);
00012 }
00013
00014 void SLLScene::update() {
00015     if (this->isMenuOpen) {
00016         this->menu->update();
00017
00018         constants::MenuLinkedList::Button status = this->menu->getActiveOptionsMenu();
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 ==
00031 constants::MenuLinkedList::CreateMode::Button::DEFINED_LIST_BUTTON) {
00032                     if (this->menu->createModeValue[1] == "None")
00033                         break;
00034                     std::vector<std::string> values;
00035                     std::string value = this->menu->createModeValue[1];
00036                     std::stringstream ss(value);
00037                     std::string token;
00038                     while (std::getline(ss, token, ',')) {
00039                         values.push_back(token);
00040                     }
00041                     this->linkedList->createLinkedList(values);
00042                 } else if (createMode == constants::MenuLinkedList::CreateMode::Button::FILE_BUTTON) {
00043                     if (this->menu->createModeValue[2] == "None")
00044                         break;
00045                     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 }
00118
00119 void SLLScene::render() {
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 }
00129
00130 SLLScene::SLLScene(sf::RenderWindow *window) : BaseScene(window) {
00131     this->init();
00132 }
00133
00134 void SLLScene::init() {
00135     this->menu = new MenuLinkedList(this->window);
00136     this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::SINGLY);
00137 }
00138
00139 void SLLScene::reset() {
00140     this->menu->resetActiveOptionsMenu();
00141 }
00142
00143 std::vector<EventAnimation> SLLScene::addModeEvents(int chosenNode) {
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     else {
00162         event.titleNodes.emplace_back(chosenNode, "temp");
00163         if (this->linkedList->getSize())
00164             event.titleNodes.emplace_back(1, "head");
00165     }
00166     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00167     if (chosenNode && chosenNode == this->linkedList->getSize())
00168         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00169     event.colorNodes.push_back(chosenNode);
00170     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00171     event.lines = {0};
00172
00173     events.emplace_back(event);
00174
00175     if (chosenNode == 0) {
00176         if (this->linkedList->getSize()) {
00177             event.reset();
00178             event.titleNodes = {
00179                 {1, "head"},
00180                 {chosenNode, "temp"}
00181             };
00182             event.colorNodes = std::vector<int>{0};
00183             event.colorArrows.emplace_back(0, NodeInfo::ArrowType::RIGHT);
00184             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00185             event.isPrintNormal = true;
00186             event.lines = {1, 2};
00187
00188             events.emplace_back(event);
00189         }
00190
00191         event.reset();
00192         event.titleNodes.emplace_back(chosenNode, "head|temp");
00193         event.lines = {3};
00194         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00195         events.emplace_back(event);
00196     } else {
00197         event.reset();
00198         event.titleNodes = {
00199             {0, "head|current"},
00200             {chosenNode, "temp"}
00201         };
00202         event.colorNodes.push_back(0);
00203         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00204         if (chosenNode == this->linkedList->getSize())
00205             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00206         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00207         event.lines = {5};
00208
00209         events.emplace_back(event);
00210
00211         for (int i = 0; i < chosenNode; ++i) {
00212             event.reset();
00213             event.titleNodes = {

```

```

00214         {0, "head"},
00215         {chosenNode, "temp"},
00216         {i, "current"}
00217     };
00218     event.colorNodes.push_back(i);
00219     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00220     if (chosenNode == this->linkedList->getSize())
00221         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00222     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00223     event.lines = {6};
00224
00225     events.emplace_back(event);
00226
00227     if (i == chosenNode - 1) break;
00228
00229     event.reset();
00230     event.titleNodes = {
00231         {0, "head"},
00232         {chosenNode, "temp"},
00233         {i, "current"}
00234     };
00235     event.colorNodes.push_back(i);
00236     event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00237     event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00238     if (chosenNode == this->linkedList->getSize())
00239         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00240     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00241     event.lines = {7};
00242
00243     events.emplace_back(event);
00244 }
00245
00246 if (chosenNode != this->linkedList->getSize()) {
00247     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 }
00275
00276 std::vector<EventAnimation> SLLScene::deleteModeEvents(int chosenNode) {
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();
00320     event.titleNodes.emplace_back(0, "head|current");
00321     event.colorNodes.push_back(0);
00322     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00323     event.lines = {5};
00324
00325     events.emplace_back(event);
00326
00327     for (int i = 0; i < chosenNode; ++i) {
00328         event.reset();
00329         event.titleNodes = {
00330             {0, "head"},
00331             {i, "current"}
00332         };
00333         event.colorNodes.push_back(i);
00334         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00335         event.lines = {6};
00336
00337         events.emplace_back(event);
00338
00339         if (i == chosenNode - 1) break;
00340
00341         event.reset();
00342         event.titleNodes = {
00343             {0, "head"},
00344             {i, "current"}
00345         };
00346         event.colorNodes.push_back(i);
00347         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00348         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00349         event.lines = {7};
00350
00351         events.emplace_back(event);
00352     }
00353
00354     event.reset();
00355     event.titleNodes = {
00356         {0, "head"},
00357         {chosenNode, "temp"},
00358         {chosenNode - 1, "current"}
00359     };
00360     event.colorNodes.push_back(chosenNode);
00361     event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00362     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00363     event.lines = {8};
00364
00365     events.emplace_back(event);
00366
00367     if (chosenNode != this->linkedList->getSize() - 1) {
00368         event.reset();
00369         event.titleNodes = {
00370             {0, "head"},
00371             {chosenNode, "temp"},
00372             {chosenNode - 1, "current"}
00373         };
00374         event.colorNodes.push_back(chosenNode);
00375         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00376         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00377         event.isPrintNormal = true;
00378         event.lines = {9};
00379
00380         events.emplace_back(event);
00381
00382         event.reset();
00383         event.titleNodes.emplace_back(0, "head");
00384         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00385         event.lines = {10};
00386
00387         events.emplace_back(event);

```

```

00388         } else {
00389             event.reset();
00390             event.titleNodes = {
00391                 {0, "head"},
00392                 {chosenNode, "temp"},
00393                 {chosenNode - 1, "current"}
00394             };
00395             event.colorNodes.push_back(chosenNode);
00396             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00397             event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00398             event.lines = {9};
00399
00400             events.emplace_back(event);
00401
00402             event.reset();
00403             event.titleNodes.emplace_back(0, "head");
00404             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00405             event.statusChosenNode = NodeInfo::StatusNode::Visible;
00406             event.lines = {10};
00407
00408             events.emplace_back(event);
00409         }
00410     }
00411     return events;
00412 }
00413 }
00414
00415 std::vector<EventAnimation> SLLScene::updateModeEvents(int chosenNode) {
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         }
00462     }
00463
00464     event.reset();
00465     if (chosenNode == 0)
00466         event.titleNodes.emplace_back(0, "head|current");
00467     else
00468         event.titleNodes = {
00469             {0, "head"},
00470             {chosenNode, "current"}
00471         };
00472     event.lines = {3};
00473
00474     events.emplace_back(event);

```

```

00475
00476     return events;
00477 }
00478
00479 std::vector<EventAnimation> SLLScene::searchModeEvents(int chosenNode) {
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 }

```

## 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.](#)

```
00001 //
00002 // Created by dirii on 26/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_SLLSCENE_HPP
00006 #define VISUALGO_CS162_SLLSCENE_HPP
00007
00008 #include "BaseScene.hpp"
00009 #include "MenuLinkedList.hpp"
00010 #include "core/LinkedList.hpp"
00011
00012 class SLLScene : public BaseScene {
00013 private:
00014     MenuLinkedList* menu;
00015     LinkedList* linkedList;
00016
00017     void init();
00018
00019 public:
00020     explicit SLLScene(sf::RenderWindow* window);
00021
00022     void reset();
00023
00024     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00025     void update() override;
00026     void render() override;
00027
00028     std::vector<EventAnimation> addModeEvents(int chosenNode);
00029     std::vector<EventAnimation> deleteModeEvents(int chosenNode);
00030     std::vector<EventAnimation> updateModeEvents(int chosenNode);
00031     std::vector<EventAnimation> searchModeEvents(int chosenNode);
00032 };
00033
00034 #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.](#)

```
00001 //
00002 // Created by dirii on 28/03/2023.
00003 //
00004
00005 #include "StackScene.hpp"
00006
00007 StackScene::StackScene(sf::RenderWindow *window) : BaseScene(window) {
00008     this->init();
00009 }
00010
00011 void StackScene::update() {
00012     if (this->isMenuOpen) {
00013         this->menu->update();
00014
00015         constants::MenuDataStructure::Button status = this->menu->getActiveOptionsMenu();
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->getActiveOptionsMenu() !=
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->resetActiveOptionsMenuOnly();
00077         this->controlMenu->reset();
00078         break;
00079     case constants::MenuDataStructure::Button::CLEAR_BUTTON:
00080         if (this->menu->getActiveOptionsMenu() !=
constants::MenuDataStructure::Button::CLEAR_BUTTON)
00081             break;
00082
00083         this->linkedList->createLinkedList(0);
00084
00085         std::cout << "Cleared " << std::endl;
00086         this->menu->resetActiveOptionsMenuOnly();
00087         this->controlMenu->reset();
00088         break;
00089     }
00090 }
00091
00092 this->controlMenu->update();
00093
00094 this->linkedList->processControlMenu(this->controlMenu->getStatus());
00095 this->linkedList->setSpeed(this->controlMenu->getSpeed());
00096
00097 this->linkedList->update();
00098 }
00099
00100 void StackScene::render() {
00101     if (this->isMenuOpen)
00102         this->menu->render();
00103
00104     if (this->isDemoCodeOpen)
00105         this->linkedList->renderHighlighter();
00106
00107     this->controlMenu->render();
00108     this->linkedList->render();
00109 }

```

```

00110
00111 void StackScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00112     if (this->isMenuOpen)
00113         this->menu->pollEvents(event, mousePosView);
00114
00115     this->controlMenu->pollEvents(event, mousePosView);
00116 }
00117
00118 void StackScene::init() {
00119     this->menu = new MenuDataStructure(this->window);
00120     this->linkedList = new LinkedList(this->window, LinkedList::TypeLinkedList::SINGLY);
00121 }
00122
00123 void StackScene::reset() {
00124     this->menu->resetActiveOptionsMenu();
00125 }
00126
00127 std::vector<EventAnimation> StackScene::pushModeEvents(int chosenNode) {
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"},
00200             {i, "current"}
00201         };
00202         event.colorNodes.push_back(i);
00203         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00204         if (chosenNode == this->linkedList->getSize())
00205             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00206         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00207         event.lines = {6};
00208
00209         events.emplace_back(event);
00210
00211         if (i == chosenNode - 1) break;
00212
00213         event.reset();
00214         event.titleNodes = {
00215             {0, "head"},
00216             {chosenNode, "temp"},
00217             {i, "current"}
00218         };
00219         event.colorNodes.push_back(i);
00220         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00221         event.hiddenArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00222         if (chosenNode == this->linkedList->getSize())
00223             event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00224         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00225         event.lines = {7};
00226
00227         events.emplace_back(event);
00228     }
00229
00230     if (chosenNode != this->linkedList->getSize()) {
00231         event.reset();
00232         event.titleNodes = {
00233             {0, "head"},
00234             {chosenNode, "temp"},
00235             {chosenNode - 1, "current"}
00236         };
00237         event.colorNodes.push_back(chosenNode);
00238         event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00239         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00240         event.isPrintNormal = true;
00241         event.lines = {8};
00242
00243         events.emplace_back(event);
00244     }
00245
00246     event.reset();
00247     event.titleNodes = {
00248         {0, "head"},
00249         {chosenNode, "temp"}
00250     };
00251     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00252     event.lines = {9};
00253
00254     events.emplace_back(event);
00255 }
00256
00257 return events;
00258 }
00259
00260 std::vector<EventAnimation> StackScene::popModeEvents(int chosenNode) {
00261     this->linkedList->resetEvents();
00262     if (chosenNode < 0 || chosenNode >= this->linkedList->getSize())
00263         return {};
00264
00265     this->linkedList->initHighlighter(
00266         constants::Highlighter::SLL::CODES_PATH[1].second,
00267         constants::Highlighter::SLL::CODES_PATH[1].first
00268     );
00269
00270     std::vector<EventAnimation> events;
00271     EventAnimation event;
00272
00273     if (!chosenNode) {
00274         event.titleNodes.emplace_back(chosenNode, "head|temp");
00275         event.colorNodes.push_back(chosenNode);
00276         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00277         event.lines = {0, 1};
00278
00279         events.emplace_back(event);
00280
00281         if (this->linkedList->getSize() > 1) {
00282             event.reset();
00283             event.titleNodes = {

```

```

00284         {chosenNode, "temp"},
00285         {1, "head"}
00286     };
00287     event.colorNodes.push_back(1);
00288     event.colorArrows.emplace_back(chosenNode, NodeInfo::ArrowType::RIGHT);
00289     event.isPrintNormal = true;
00290     event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00291     event.lines = {2};
00292
00293     events.emplace_back(event);
00294 }
00295
00296 event.reset();
00297 event.titleNodes.emplace_back(1, "head");
00298 event.statusChosenNode = NodeInfo::StatusNode::Visible;
00299 event.lines = {3};
00300
00301 events.emplace_back(event);
00302 } else {
00303     event.reset();
00304     event.titleNodes.emplace_back(0, "head|current");
00305     event.colorNodes.push_back(0);
00306     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00307     event.lines = {5};
00308
00309     events.emplace_back(event);
00310
00311     for (int i = 0; i < chosenNode; ++i) {
00312         event.reset();
00313         event.titleNodes = {
00314             {0, "head"},
00315             {i, "current"}
00316         };
00317         event.colorNodes.push_back(i);
00318         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00319         event.lines = {6};
00320
00321         events.emplace_back(event);
00322
00323         if (i == chosenNode - 1) break;
00324
00325         event.reset();
00326         event.titleNodes = {
00327             {0, "head"},
00328             {i, "current"}
00329         };
00330         event.colorNodes.push_back(i);
00331         event.colorArrows.emplace_back(i, NodeInfo::ArrowType::RIGHT);
00332         event.statusChosenNode = NodeInfo::StatusNode::InChain;
00333         event.lines = {7};
00334
00335         events.emplace_back(event);
00336     }
00337
00338     event.reset();
00339     event.titleNodes = {
00340         {0, "head"},
00341         {chosenNode, "temp"},
00342         {chosenNode - 1, "current"}
00343     };
00344     event.colorNodes.push_back(chosenNode);
00345     event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00346     event.statusChosenNode = NodeInfo::StatusNode::InChain;
00347     event.lines = {8};
00348
00349     events.emplace_back(event);
00350
00351     if (chosenNode != this->linkedList->getSize() - 1) {
00352         event.reset();
00353         event.titleNodes = {
00354             {0, "head"},
00355             {chosenNode, "temp"},
00356             {chosenNode - 1, "current"}
00357         };
00358         event.colorNodes.push_back(chosenNode);
00359         event.colorArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00360         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00361         event.isPrintNormal = true;
00362         event.lines = {9};
00363
00364         events.emplace_back(event);
00365
00366         event.reset();
00367         event.titleNodes.emplace_back(0, "head");
00368         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00369         event.lines = {10};
00370

```



```

00371         events.emplace_back(event);
00372     } else {
00373         event.reset();
00374         event.titleNodes = {
00375             {0, "head"},
00376             {chosenNode, "temp"},
00377             {chosenNode - 1, "current"}
00378         };
00379         event.colorNodes.push_back(chosenNode);
00380         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00381         event.statusChosenNode = NodeInfo::StatusNode::OutChain;
00382         event.lines = {9};
00383
00384         events.emplace_back(event);
00385
00386         event.reset();
00387         event.titleNodes.emplace_back(0, "head");
00388         event.hiddenArrows.emplace_back(chosenNode - 1, NodeInfo::ArrowType::RIGHT);
00389         event.statusChosenNode = NodeInfo::StatusNode::Visible;
00390         event.lines = {10};
00391
00392         events.emplace_back(event);
00393     }
00394 }
00395
00396 return events;
00397 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 28/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_STACKSCENE_HPP
00006 #define VISUALGO_CS162_STACKSCENE_HPP
00007
00008 #include "BaseScene.hpp"
00009 #include "MenuDataStructure.hpp"
00010 #include "core/LinkedList.hpp"
00011
00012 class StackScene : public BaseScene{
00013 private:
00014     MenuDataStructure* menu;
00015     LinkedList* linkedList;
00016
00017     void init();
00018
00019 public:
00020     explicit StackScene(sf::RenderWindow* window);
00021
00022     void reset();
00023
00024     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00025     void update() override;
00026     void render() override;
00027
00028     std::vector<EventAnimation> pushModeEvents(int chosenNode);
00029     std::vector<EventAnimation> popModeEvents(int chosenNode);
00030 };
00031
00032 #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.](#)

```
00001 //
00002 // Created by dirii on 27/03/2023.
00003 //
00004
00005 #include "StaticArrayScene.hpp"
00006
00007 StaticArrayScene::StaticArrayScene(sf::RenderWindow *window) : BaseScene(window) {
00008     this->init();
00009 }
00010
00011 void StaticArrayScene::update() {
00012     if (this->isMenuOpen) {
00013         this->menu->update();
00014
00015         constants::MenuArray::Button status = this->menu->getActiveOptionsMenu();
00016         constants::MenuArray::CreateMode::Button createMode;
00017         switch (status){
00018             case constants::MenuArray::Button::CREATE_BUTTON:
00019                 createMode = this->menu->getActiveCreateMode();
00020                 if (createMode == constants::MenuArray::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->array->createArray(size);
00027                 } else if (createMode ==
00028 constants::MenuArray::CreateMode::Button::DEFINED_LIST_BUTTON) {
00029                     if (this->menu->createModeValue[1] == "None")
00030                         break;
00031                     std::vector<std::string> values;
00032                     std::string value = this->menu->createModeValue[1];
00033                     std::stringstream ss(value);
00034                     std::string token;
00035                     while (std::getline(ss, token, ',')) {
00036                         values.push_back(token);
00037                     }
00038                     this->array->createArray(values);
00039                 } else if (createMode == constants::MenuArray::CreateMode::Button::FILE_BUTTON) {
00040                     if (this->menu->createModeValue[2] == "None")
00041                         break;
00042                     std::vector<std::string> values;
00043                     std::string value = this->menu->createModeValue[2];
00044                     std::stringstream ss(value);
00045                     std::string token;
00046                     while (std::getline(ss, token, ','))
00047                         values.push_back(token);
00048                     this->array->createArray(values);
00049                     this->menu->createModeValue[2] = "None";
00050                 }
00051                 this->controlMenu->reset();
00052                 break;
00053             case constants::MenuArray::Button::ADD_BUTTON:
00054                 if (this->menu->addModeValue[0] == "None" || this->menu->addModeValue[1] == "None" ||
00055 this->menu->addModeValue[0].empty())
00056                     break;
00057                 this->array->addSquare(
00058                     std::stoi(this->menu->addModeValue[0]),
00059                     this->menu->addModeValue[1],
00060                     this->addModeEvents(std::stoi(this->menu->addModeValue[0]))
00061                 );
00062                 std::cout << "Add: " << this->menu->addModeValue[0] << " " << this->menu->addModeValue[1]
00063 << std::endl;
00064                 this->menu->addModeValue[0] = this->menu->addModeValue[1] = "None";
00065                 this->controlMenu->reset();
00066                 break;
00067             case constants::MenuArray::Button::DELETE_BUTTON:
00068                 if (this->menu->deleteModeValue == "None" || this->menu->deleteModeValue.empty())
```

```

00068         break;
00069
00070         this->array->deleteSquare(
00071             std::stoi(this->menu->deleteModeValue),
00072             this->deleteModeEvents(std::stoi(this->menu->deleteModeValue))
00073         );
00074
00075         std::cout << "Delete: " << this->menu->deleteModeValue << std::endl;
00076         this->menu->deleteModeValue = "None";
00077         this->controlMenu->reset();
00078         break;
00079     case constants::MenuArray::Button::UPDATE_BUTTON:
00080         if (this->menu->updateModeValue[0] == "None" || this->menu->updateModeValue[1] ==
00081             "None" || this->menu->updateModeValue[0].empty())
00082             break;
00083
00084         this->array->updateSquare(
00085             std::stoi(this->menu->updateModeValue[0]),
00086             this->menu->updateModeValue[1],
00087             this->updateModeEvents(std::stoi(this->menu->updateModeValue[0]))
00088         );
00089
00090         std::cout << "Update: " << this->menu->updateModeValue[0] << " " <<
00091             this->menu->updateModeValue[1] << std::endl;
00092         this->menu->updateModeValue[0] = this->menu->updateModeValue[1] = "None";
00093         this->controlMenu->reset();
00094         break;
00095     case constants::MenuArray::Button::SEARCH_BUTTON:
00096         if (this->menu->searchModeValue == "None" || this->menu->searchModeValue.empty())
00097             break;
00098
00099         this->array->searchSquare(
00100             this->searchModeEvents(this->array->findValue(this->menu->searchModeValue))
00101         );
00102
00103         std::cout << "Search: " << this->menu->searchModeValue << std::endl;
00104         this->menu->searchModeValue = "None";
00105         this->controlMenu->reset();
00106         break;
00107     }
00108 }
00109
00110 this->controlMenu->update();
00111
00112 this->array->processControlMenu(this->controlMenu->getStatus());
00113 this->array->setSpeed(this->controlMenu->getSpeed());
00114
00115 this->array->update();
00116 }
00117
00118 void StaticArrayScene::render() {
00119     if (this->isMenuOpen)
00120         this->menu->render();
00121
00122     if (this->isDemoCodeOpen)
00123         this->array->renderHighlighter();
00124
00125     this->controlMenu->render();
00126     this->array->render();
00127 }
00128
00129 void StaticArrayScene::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00130     if (this->isMenuOpen)
00131         this->menu->pollEvents(event, mousePosView);
00132
00133     this->controlMenu->pollEvents(event, mousePosView);
00134 }
00135
00136 void StaticArrayScene::init() {
00137     this->menu = new MenuArray(this->window, constants::MenuArray::Type::STATIC);
00138     this->array = new Array(this->window, Array::TypeArray::STATIC);
00139 }
00140
00141 void StaticArrayScene::reset() {
00142     this->menu->resetActiveOptionsMenu();
00143 }
00144
00145 std::vector<EventAnimation> StaticArrayScene::addModeEvents(int chosenNode) {
00146     this->array->resetEvents();
00147     if (chosenNode < 0 || chosenNode > this->array->getSize())
00148         return {};
00149
00150     // init highlighter
00151     // ...
00152
00153     int size = this->array->getSize() + 1,
00154         squaresSize = this->array->getSquaresSize();

```

```

00153     if (size > squaresSize) --size;
00154     if (!squaresSize) return {};
00155
00156     std::vector<EventAnimation> events;
00157     EventAnimation event;
00158
00159     if (size < squaresSize) {
00160         event = EventAnimation();
00161         event.eventSquares.assign(squaresSize, EventSquare());
00162         for (auto &square: event.eventSquares) {
00163             square.status = Square::Status::active;
00164             square.isPrintPreVal = true;
00165         }
00166         for (int i = size - 1; i < squaresSize; ++i)
00167             event.eventSquares[i].status = Square::Status::inactive;
00168         if (size > 1)
00169             event.eventSquares[size - 2].title = "n";
00170
00171         events.emplace_back(event);
00172
00173         event = EventAnimation();
00174         event.eventSquares.assign(squaresSize, EventSquare());
00175         for (auto &square : event.eventSquares) {
00176             square.status = Square::Status::active;
00177             square.isPrintPreVal = true;
00178         }
00179         for (int i = size; i < squaresSize; ++i)
00180             event.eventSquares[i].status = Square::Status::inactive;
00181         event.eventSquares[size - 1].title = "n";
00182
00183         events.emplace_back(event);
00184     }
00185
00186     for (int i = size - 1; i >= chosenNode; --i) {
00187         event = EventAnimation();
00188         event.eventSquares.assign(squaresSize, EventSquare());
00189         for (auto &square: event.eventSquares) {
00190             square.status = Square::Status::active;
00191             square.isPrintPreVal = true;
00192         }
00193         for (int j = size; j < squaresSize; ++j)
00194             event.eventSquares[j].status = Square::Status::inactive;
00195         event.eventSquares[size - 1].title = "n";
00196         for (int j = size - 1; j > i; --j)
00197             event.eventSquares[j].isPrintPreVal = false;
00198         event.eventSquares[i].status = Square::Status::chosen;
00199
00200         events.emplace_back(event);
00201
00202         event.eventSquares[i].isPrintPreVal = false;
00203         if (i > chosenNode)
00204             event.eventSquares[i - 1].status = Square::Status::chosen;
00205
00206         events.emplace_back(event);
00207     }
00208
00209     return events;
00210 }
00211
00212 std::vector<EventAnimation> StaticArrayScene::deleteModeEvents(int chosenNode) {
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 }
00264
00265 std::vector<EventAnimation> StaticArrayScene::updateModeEvents(int chosenNode) {
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 }
00294
00295 std::vector<EventAnimation> StaticArrayScene::searchModeEvents(int chosenNode) {
00296     this->array->resetEvents();
00297
00298     // init highlighter
00299     // ...
00300
00301     int size = this->array->getSize(),
00302         squaresSize = this->array->getSquaresSize();
00303     std::vector<EventAnimation> events;
00304     EventAnimation event;
00305
00306     for (int i = 0; i <= chosenNode; ++i) {
00307         if (i == size) break;
00308
00309         event = EventAnimation();
00310         event.eventSquares.assign(squaresSize, EventSquare());
00311         for (int j = 0; j < size; ++j) {
00312             event.eventSquares[j].status = Square::Status::active;
00313             if (j == size - 1)
00314                 event.eventSquares[size - 1].title = "n";
00315         }
00316         event.eventSquares[i].status = Square::Status::chosen;
00317
00318         events.emplace_back(event);
00319     }
00320
00321     if (chosenNode == size) {
00322         event = EventAnimation();
00323         event.eventSquares.assign(squaresSize, EventSquare());
00324         for (int j = 0; j < size; ++j) {
00325             event.eventSquares[j].status = Square::Status::active;
00326             if (j == size - 1)

```

```

00327         event.eventSquares[size - 1].title = "n";
00328     }
00329
00330     events.emplace_back(event);
00331 }
00332
00333 return events;
00334 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 27/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_STATICARRAYSCENE_HPP
00006 #define VISUALGO_CS162_STATICARRAYSCENE_HPP
00007
00008 #include "BaseScene.hpp"
00009 #include "MenuArray.hpp"
00010 #include "core/Array.hpp"
00011
00012 class StaticArrayScene : public BaseScene{
00013 private:
00014     MenuArray* menu;
00015     Array* array;
00016
00017     void init();
00018
00019 public:
00020     explicit StaticArrayScene(sf::RenderWindow* window);
00021
00022     void reset();
00023
00024     void pollEvent(sf::Event event, sf::Vector2f mousePosView) override;
00025     void update() override;
00026     void render() override;
00027
00028     std::vector<EventAnimation> addModeEvents(int chosenNode);
00029     std::vector<EventAnimation> deleteModeEvents(int chosenNode);
00030     std::vector<EventAnimation> updateModeEvents(int chosenNode);
00031     std::vector<EventAnimation> searchModeEvents(int chosenNode);
00032 };
00033
00034 #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.](#)

```
00001 //
00002 // Created by dirii on 23/03/2023.
00003 //
00004
00005 #include "MousePosition.hpp"
00006
00007 void MousePosition::updateMousePosition() {
00008     this->mousePos = sf::Mouse::getPosition(*this->relativeWindow);
00009     this->mousePosView = this->relativeWindow->mapPixelToCoords(this->mousePos);
00010 }
```

## 8.109 include/MousePosition.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

### Classes

- class [MousePosition](#)

## 8.110 MousePosition.hpp

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by dirii on 23/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_MOUSEPOSITION_HPP
00006 #define VISUALGO_CS162_MOUSEPOSITION_HPP
00007
00008 #include <SFML/Graphics.hpp>
00009
00010 class MousePosition{
00011 protected:
00012     sf::RenderWindow* relativeWindow;
00013
00014     sf::Vector2i mousePos;
00015     sf::Vector2f mousePosView;
00016 public:
00017     void updateMousePosition();
00018 };
00019
00020 #endif //VISUALGO_CS162_MOUSEPOSITION_HPP
```

## 8.111 include/stuff/button.cpp File Reference

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

## 8.112 button.cpp

[Go to the documentation of this file.](#)

```

00001 //
00002 // Created by dirii on 25/03/2023.
00003 //
00004
00005 #include "button.hpp"
00006
00007 #include <utility>
00008
00009 void Button::init() {
00010     this->isHover = this->isClick = false;
00011
00012     this->button.setSize(this->size);
00013     this->button.setFillColor(this->color);
00014     this->button.setPosition(this->position);
00015
00016     // set rounded corners
00017     this->button.setCornerPointCount(constants::CORNER_POINT_COUNT_BUTTON);
00018     this->button.setCornersRadius(constants::CORNER_RADIUS_BUTTON);
00019
00020
00021     this->text.setFont(this->font);
00022     this->text.setString(this->textString);
00023     this->text.setCharacterSize(this->textSize);
00024     this->text.setFillColor(this->textColor);
00025     this->text.setPosition(this->position.x + this->size.x / 2.0 - this->text.getGlobalBounds().width
/ 2.0,
00026                             this->position.y + this->size.y / 2.0 - this->text.getGlobalBounds().height
/ 1.1);
00027 }
00028
00029 Button::Button(sf::RenderWindow *window, sf::Vector2f position, sf::Vector2f size,
00030               std::string textString, std::string changedTextString, int textSize,
00031               sf::Color textColor, sf::Color color, sf::Color hoverColor, sf::Color clickColor) {
00032     this->window = window;
00033     this->position = position;
00034     this->size = size;
00035     this->textString = std::move(textString);
00036     this->changedTextString = std::move(changedTextString);
00037     this->textSize = textSize;
00038     this->color = color;
00039     this->textColor = textColor;
00040     this->hoverColor = hoverColor;
00041     this->clickColor = clickColor;
00042     this->font.loadFromFile(constants::fontPath);
00043
00044     this->init();
00045 }
00046
00047 bool Button::pollEvent(sf::Vector2f mousePosView) {
00048     bool hasClicked = false;
00049
00050     if (this->isHover and this->isClick and !sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
00051         hasClicked = true;
00052         std::swap(this->textString, this->changedTextString);
00053     }
00054
00055     this->isHover = this->button.getGlobalBounds().contains(mousePosView);
00056     this->isClick = sf::Mouse::isButtonPressed(sf::Mouse::Left);
00057
00058     return hasClicked;
00059 }
00060
00061 void Button::update() {
00062     if (this->isHover) {
00063         this->button.setFillColor(this->hoverColor);
00064
00065         if (this->isClick) {
00066             this->button.setFillColor(this->clickColor);
00067         }
00068     } else {
00069         this->button.setFillColor(this->color);
00070     }
00071
00072     this->text.setString(this->textString);
00073 }
00074
00075 void Button::render() {
00076     this->window->draw(this->button);
00077     this->window->draw(this->text);
00078 }
00079
00080 bool Button::checkClicked() const {

```



```

00081     return this->isClick and this->isHover;
00082 }
00083
00084 std::string Button::getTextString() const {
00085     return this->textString;
00086 }
00087
00088 void Button::setColor(sf::Color _color) {
00089     this->color = _color;
00090 }
00091
00092 Button::Button() {}
00093
00094 sf::Vector2f Button::getPosition() const {
00095     return this->position;
00096 }
00097
00098 sf::Vector2f Button::getSize() const {
00099     return this->size;
00100 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 25/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_BUTTON_HPP
00006 #define VISUALGO_CS162_BUTTON_HPP
00007
00008 #include <SFML/Graphics.hpp>
00009 #include "RoundedRectangleShape.hpp"
00010 #include "Constants.hpp"
00011
00012 class Button{
00013 private:
00014     sf::RenderWindow* window;
00015     sf::RoundedRectangleShape button;
00016     sf::Text text;
00017     sf::Font font;
00018     sf::Color color;
00019     sf::Color textColor;
00020     sf::Color hoverColor;
00021     sf::Color clickColor;
00022     sf::Vector2f position;
00023     sf::Vector2f size;
00024     int textSize;
00025     std::string textString;
00026     std::string changedTextString;
00027     bool isHover;
00028     bool isClick;
00029
00030     void init();
00031
00032 public:
00033     Button();
00034     Button(
00035         sf::RenderWindow* window,

```

```

00036         sf::Vector2f position,
00037         sf::Vector2f size,
00038         std::string textString,
00039         std::string changedTextString,
00040         int textSize,
00041         sf::Color textColor,
00042         sf::Color color,
00043         sf::Color hoverColor,
00044         sf::Color clickColor
00045     );
00046
00047     bool pollEvent(sf::Vector2f mousePosView);
00048     void update();
00049     void render();
00050
00051     void setColor(sf::Color _color);
00052     std::string getTextString() const;
00053     sf::Vector2f getPosition() const;
00054     sf::Vector2f getSize() const;
00055
00056     bool checkClicked() const;
00057 };
00058
00059 #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.](#)

```

00001 //
00002 // Created by dirii on 04/04/2023.
00003 //
00004
00005 #include "CustomTextbox.hpp"
00006
00007 CustomTextbox::CustomTextbox(sf::RenderWindow *window, sf::Vector2f position, int size,
00008                             std::string _titleString, int maxLength) {
00009     this->window = window;
00010     this->titleString = std::move(_titleString);
00011     this->position = position;
00012
00013     font.loadFromFile(constants::fontPath);
00014     this->title.setFont(font);
00015     this->title.setCharacterSize(size);
00016     this->title.setFillColor(sf::Color::Black);
00017     this->title.setString(this->titleString);
00018     this->title.setPosition(this->position);
00019
00020     float width = this->title.findCharacterPos(this->titleString.size() - 1).x -
00021                 this->title.findCharacterPos(0).x;
00022     // std::cout << width << ' ' << this->title.getString().getSize() << std::endl;
00023
00024     this->maxLength = maxLength;
00025
00026     this->textbox = new TextBox(
00027         this->window,
00028         sf::Vector2f(this->position.x + width + 10, this->position.y),
00029         20,
00030         sf::Color::Black,
00031         sf::Color::White,
00032         this->maxLength
00033     );
00034
00035     this->goButton = new Button(
00036         this->window,
00037         sf::Vector2f(this->textbox->getBox().getPosition().x + this->textbox->getBox().getSize().x +
00038                     10, this->position.y),
00038         constants::goButtonSize,
00039         "Go",
00040         "Go",

```

```

00041         20,
00042         sf::Color::Black,
00043         constants::normalGray,
00044         constants::hoverGray,
00045         constants::clickGray
00046     );
00047
00048     this->isGoButtonClicked = false;
00049 }
00050
00051 void CustomTextbox::pollEvent(sf::Event event, sf::Vector2f mousePosView) {
00052     this->textbox->pollEvent(event);
00053     if (this->goButton->pollEvent(mousePosView)) {
00054         this->isGoButtonClicked = true;
00055         //         std::cout << "Go button clicked!\n";
00056     }
00057 }
00058
00059 void CustomTextbox::update() {
00060     this->textbox->update();
00061     this->goButton->update();
00062 }
00063
00064 void CustomTextbox::render() {
00065     this->window->draw(this->title);
00066     this->textbox->render();
00067     this->goButton->render();
00068 }
00069
00070 std::string CustomTextbox::getTextString() {
00071     if (this->isGoButtonClicked) {
00072         this->isGoButtonClicked = false;
00073         return this->textbox->getTextString();
00074     }
00075     return "None";
00076 }
00077
00078 void CustomTextbox::resetInput() {
00079     this->textbox->resetInput();
00080 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 04/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_CUSTOMTEXTBOX_HPP
00006 #define VISUALGO_CS162_CUSTOMTEXTBOX_HPP
00007
00008 #include "Textbox.hpp"
00009 #include "button.hpp"
00010
00011 class CustomTextbox{
00012 private:
00013     sf::RenderWindow* window;
00014     sf::Vector2f position;
00015     Button* goButton;
00016     TextBox* textbox;

```

```

00017     sf::Font font;
00018     sf::Text title;
00019     std::string titleString;
00020     int maxLength;
00021     bool isGoButtonClicked;
00022
00023 public:
00024     CustomTextbox(sf::RenderWindow* window, sf::Vector2f position, int size, std::string titleString,
00025         int maxLength);
00026     ~CustomTextbox() = default;
00027     void pollEvent(sf::Event event, sf::Vector2f mousePosView);
00028     void update();
00029     void render();
00030
00031     std::string getTextString();
00032     void resetInput();
00033 };
00034
00035 #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.](#)

```

00001
00002 //
00003 // This software is provided 'as-is', without any express or implied warranty.
00004 // In no event will the authors be held liable for any damages arising from the use of this software.
00005 //
00006 // Permission is granted to anyone to use this software for any purpose,
00007 // including commercial applications, and to alter it and redistribute it freely,
00008 // subject to the following restrictions:
00009 //
00010 // 1. The origin of this software must not be misrepresented;
00011 // you must not claim that you wrote the original software.
00012 // If you use this software in a product, an acknowledgment
00013 // in the product documentation would be appreciated but is not required.
00014 //
00015 // 2. Altered source versions must be plainly marked as such,
00016 // and must not be misrepresented as being the original software.
00017 //
00018 // 3. This notice may not be removed or altered from any source distribution.
00019 //
00020
00021 // Headers
00022 #include "RoundedRectangleShape.hpp"
00023 #include <cmath>
00024
00025 namespace sf
00026 {
00027     RoundedRectangleShape::RoundedRectangleShape(const Vector2f& size, float radius, unsigned int
00028         cornerPointCount)
00029     {
00030         mySize = size;
00031         myRadius = radius;
00032         myCornerPointCount = cornerPointCount;
00033         update();
00034     }
00035
00036     void RoundedRectangleShape::setSize(const Vector2f& size)

```

```

00041     {
00042         mySize = size;
00043         update();
00044     }
00045
00047     const Vector2f& RoundedRectangleShape::getSize() const
00048     {
00049         return mySize;
00050     }
00051
00053     void RoundedRectangleShape::setCornersRadius(float radius)
00054     {
00055         myRadius = radius;
00056         update();
00057     }
00058
00060     float RoundedRectangleShape::getCornersRadius() const
00061     {
00062         return myRadius;
00063     }
00064
00066     void RoundedRectangleShape::setCornerPointCount(unsigned int count)
00067     {
00068         myCornerPointCount = count;
00069         update();
00070     }
00071
00073     std::size_t RoundedRectangleShape::getPointCount() const
00074     {
00075         return myCornerPointCount*4;
00076     }
00077
00079     sf::Vector2f RoundedRectangleShape::getPoint(std::size_t index) const
00080     {
00081         if(index >= myCornerPointCount*4)
00082             return sf::Vector2f(0,0);
00083
00084         float deltaAngle = 90.0f/(myCornerPointCount-1);
00085         sf::Vector2f center;
00086         unsigned int centerIndex = index/myCornerPointCount;
00087         static const float pi = 3.141592654f;
00088
00089         switch(centerIndex)
00090         {
00091             case 0: center.x = mySize.x - myRadius; center.y = myRadius; break;
00092             case 1: center.x = myRadius; center.y = myRadius; break;
00093             case 2: center.x = myRadius; center.y = mySize.y - myRadius; break;
00094             case 3: center.x = mySize.x - myRadius; center.y = mySize.y - myRadius; break;
00095         }
00096
00097         return sf::Vector2f(myRadius*cos(deltaAngle*(index-centerIndex)*pi/180)+center.x,
00098                             -myRadius*sin(deltaAngle*(index-centerIndex)*pi/180)+center.y);
00099     }
00100 } // 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.](#)

```

00001
00002 //
00003 // This software is provided 'as-is', without any express or implied warranty.
00004 // In no event will the authors be held liable for any damages arising from the use of this software.
00005 //
00006 // Permission is granted to anyone to use this software for any purpose,
00007 // including commercial applications, and to alter it and redistribute it freely,
00008 // subject to the following restrictions:
00009 //
00010 // 1. The origin of this software must not be misrepresented;
00011 // you must not claim that you wrote the original software.
00012 // If you use this software in a product, an acknowledgment
00013 // in the product documentation would be appreciated but is not required.
00014 //
00015 // 2. Altered source versions must be plainly marked as such,
00016 // and must not be misrepresented as being the original software.
00017 //
00018 // 3. This notice may not be removed or altered from any source distribution.
00019 //
00020
00021
00022 #ifndef ROUNDEDRECTANGLESHAPE_HPP
00023 #define ROUNDEDRECTANGLESHAPE_HPP
00024
00025 // Headers
00026 #include <SFML/Graphics/Shape.hpp>
00027
00028 namespace sf
00029 {
00030     class RoundedRectangleShape : public Shape
00031     {
00032     public:
00033         explicit RoundedRectangleShape(const Vector2f& size = Vector2f(0, 0), float radius = 0,
00034             unsigned int cornerPointCount = 0);
00035
00036         void setSize(const Vector2f& size);
00037
00038         const Vector2f& getSize() const;
00039
00040         void setCornersRadius(float radius);
00041
00042         float getCornersRadius() const;
00043
00044         void setCornerPointCount(unsigned int count);
00045
00046         virtual std::size_t getPointCount() const;
00047
00048         virtual sf::Vector2f getPoint(std::size_t index) const;
00049
00050     private:
00051         // Member data
00052         Vector2f mySize;
00053         float myRadius;
00054         unsigned int myCornerPointCount;
00055     };
00056 }
00057 #endif // ROUNDEDRECTANGLESHAPE_HPP
00058
00059
00060

```

## 8.123 include/stuff/Textbox.cpp File Reference

```
#include "Textbox.hpp"
```

## 8.124 Textbox.cpp

[Go to the documentation of this file.](#)

```

00001 //
00002 // Created by dirii on 01/04/2023.

```

```

00003 //
00004
00005 #include "Textbox.hpp"
00006
00007 TextBox::TextBox(sf::RenderWindow* window, sf::Vector2f position, int size, sf::Color textColor,
00008                 sf::Color boxColor, int maxLength) {
00009     this->window = window;
00010
00011     this->cursor = "|";
00012
00013     this->box.setPosition(position);
00014     this->box.setSize(sf::Vector2f(static_cast<float>((maxLength + 1) * 12), static_cast<float>(size *
1.5)));
00015     this->box.setFillColor(boxColor);
00016     this->box.setOutlineColor(sf::Color::Black);
00017     this->box.setOutlineThickness(1);
00018
00019     this->font.loadFromFile(constants::fontPath);
00020     this->text.setFont(this->font);
00021     this->text.setCharacterSize(size);
00022     this->text.setFillColor(textColor);
00023     this->text.setPosition(position);
00024
00025     this->maxLength = maxLength;
00026     this->textColor = textColor;
00027     this->boxColor = boxColor;
00028
00029     this->cursorVisible = true;
00030     this->flickerClock.restart();
00031 }
00032
00033 void TextBox::pollEvent(sf::Event event) {
00034     if (event.type == sf::Event::TextEntered)
00035     {
00036         if (event.text.unicode == '\b')
00037         {
00038             if (!this->inputString.empty())
00039             {
00040                 this->inputString.pop_back();
00041             }
00042         }
00043         else if (((48 <= event.text.unicode && event.text.unicode <= 57) || event.text.unicode ==
static_cast<int>(',')) && this->inputString.size() < this->maxLength)
00044         {
00045             this->inputString += static_cast<char>(event.text.unicode);
00046         }
00047         this->text.setString(this->inputString);
00048     }
00049
00050
00051     // if (event.type == sf::Event::Resized)
00052     // {
00053     //     box.setPosition(
00054     //         static_cast<float>(this->window->getSize().x) / 2 - box.getSize().x / 2,
00055     //         static_cast<float>(this->window->getSize().y) / 2 - box.getSize().y / 2
00056     //     );
00057     //     text.setPosition(box.getPosition().x + 10, box.getPosition().y);
00058     //     cursor.setPosition(text.getGlobalBounds().width + text.getPosition().x,
cursor.getPosition().y);
00059     // }
00060 }
00061
00062 void TextBox::update() {
00063     if (this->flickerClock.getElapsedTime().asSeconds() >= 0.5)
00064     {
00065         this->cursorVisible = !this->cursorVisible;
00066         this->flickerClock.restart();
00067     }
00068
00069     if (this->cursorVisible)
00070     {
00071         this->text.setString(this->inputString + this->cursor);
00072     }
00073     else
00074     {
00075         this->text.setString(this->inputString);
00076     }
00077 }
00078
00079 void TextBox::render() {
00080     this->window->draw(this->box);
00081     this->window->draw(this->text);
00082 }
00083
00084 std::string TextBox::getTextString() const {
00085     return this->inputString;
00086 }

```

```

00087
00088 sf::RectangleShape TextBox::getBox() const {
00089     return this->box;
00090 }
00091
00092 void TextBox::resetInput() {
00093     this->inputString = "";
00094     this->text.setString(this->inputString);
00095 }

```

## 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.](#)

```

00001 //
00002 // Created by dirii on 01/04/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_TEXTBOX_HPP
00006 #define VISUALGO_CS162_TEXTBOX_HPP
00007
00008 #include "Constants.hpp"
00009 #include <SFML/Graphics.hpp>
00010 #include <iostream>
00011 #include <string>
00012
00013 class TextBox {
00014 public:
00015     TextBox(sf::RenderWindow* window, sf::Vector2f position, int size, sf::Color textColor, sf::Color
    boxColor, int maxLength);
00016
00017     void pollEvent(sf::Event event);
00018     void update();
00019     void render();
00020
00021     std::string getTextString() const;
00022     sf::RectangleShape getBox() const;
00023     void resetInput();
00024
00025 private:
00026     sf::RenderWindow* window;
00027
00028     std::string cursor;
00029     sf::RectangleShape box;
00030     sf::Font font;
00031     sf::Text text;
00032
00033     std::string inputString;
00034
00035     int maxLength;
00036     sf::Color textColor;
00037     sf::Color boxColor;
00038
00039     bool cursorVisible;
00040     sf::Clock flickerClock;
00041 };
00042
00043 #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.](#)

```

00001 //
00002 // Created by dirii on 23/03/2023.
00003 //
00004
00005 #include "Window.hpp"
00006
00007 void Window::init() {
00008     this->relativeWindow = this->window;
00009     this->currentScene = constants::sceneVariables::MAIN_MENU_SCENE;
00010
00011     // init buttons
00012     this->submenuButton = new Button(
00013         this->window,
00014         constants::submenuButtonPos,
00015         constants::sideButtonSize,
00016         ">",
00017         "<",
00018         15,
00019         sf::Color::Black,
00020         constants::normalGray,
00021         constants::hoverGray,
00022         constants::clickGray
00023     );
00024
00025     this->demoCodeButton = new Button(
00026         this->window,
00027         constants::demoCodeButtonPos,
00028         constants::sideButtonSize,
00029         "<",
00030         ">",
00031         15,
00032         sf::Color::Black,
00033         constants::normalGray,
00034         constants::hoverGray,
00035         constants::clickGray
00036     );
00037 }
00038
00039 void Window::initWindow() {
00040     this->videoMode.width = constants::Width;
00041     this->videoMode.height = constants::Height;
00042     this->window = new sf::RenderWindow(
00043         this->videoMode,
00044         constants::titleWindow,
00045         sf::Style::Titlebar | sf::Style::Close);
00046
00047     this->window->setFramerateLimit(constants::fps);
00048 }
00049
00050 Window::Window() {
00051     this->initWindow();
00052     this->initScenes();
00053     this->init();
00054 }
00055
00056 const bool Window::running() const {
00057     return this->window->isOpen();
00058 }
00059
00060 void Window::pollEvent() {
00061     // event polling
00062     while (this->window->pollEvent(this->event)) {
00063         switch (this->event.type) {
00064             case sf::Event::Closed:
00065                 this->window->close();
00066                 break;
00067             case sf::Event::KeyPressed:
00068                 if (this->event.key.code == sf::Keyboard::Q) {
00069                     std::cout << "You have pressed Q!\n";
00070                 }
00071                 if (this->event.key.code == sf::Keyboard::W) {
00072                     std::cout << "You have pressed W!\n";
00073                 }
00074                 break;
00075             default:
00076                 break;
00077         }
00078
00079         if (this->submenuButton->pollEvent(this->mousePosView)) {
00080             std::cout << "You have clicked on submenu button!\n";
00081             this->scenes[this->currentScene]->isMenuOpen = (this->submenuButton->getTextString() ==
    "<");

```

```

00082     }
00083
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 }
00101
00102 void Window::update() {
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 }
00118
00119 void Window::render() {
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 }
00139
00140 void Window::initScenes() {
00141     this->scenes[constants::sceneVariables::MAIN_MENU_SCENE] = new MainMenu(this->window);
00142     this->scenes[constants::sceneVariables::SINGLY_LINKED_LIST_SCENE] = new SLLScene(this->window);
00143     this->scenes[constants::sceneVariables::DOUBLY_LINKED_LIST_SCENE] = new DLLScene(this->window);
00144     this->scenes[constants::sceneVariables::CIRCULAR_LINKED_LIST_SCENE] = new CLLScene(this->window);
00145     this->scenes[constants::sceneVariables::STACK_SCENE] = new StackScene(this->window);
00146     this->scenes[constants::sceneVariables::QUEUE_SCENE] = new QueueScene(this->window);
00147     this->scenes[constants::sceneVariables::STATIC_ARRAY_SCENE] = new StaticArrayScene(this->window);
00148     this->scenes[constants::sceneVariables::DYNAMIC_ARRAY_SCENE] = new
DynamicArrayScene(this->window);
00149
00150     for (int i = 1; i < constants::sceneVariables::SCENE_COUNT; i++) {
00151         this->scenes[i]->createModeButton(
00152             sf::Vector2f(
00153                 constants::modeButtonPos.x * static_cast<float>(i) +
00154                 (constants::distance2ModeButtons + constants::modeButtonSize.x) *
static_cast<float>(i - 1),
00155                 constants::modeButtonPos.y
00156             ),
00157             constants::sceneVariables::NAME_MODE_BUTTON[i]
00158         );
00159     }
00160 }

```

## 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.](#)

```
00001 //
00002 // Created by dirii on 23/03/2023.
00003 //
00004
00005 #ifndef VISUALGO_CS162_WINDOW_HPP
00006 #define VISUALGO_CS162_WINDOW_HPP
00007
00008 #include <iostream>
00009 #include <SFML/Graphics.hpp>
00010 #include "MousePosition.hpp"
00011 #include "Constants.hpp"
00012 #include "stuff/button.hpp"
00013 #include "libScene/AllScenes.hpp"
00014
00015 class Window : public MousePosition{
00016 private:
00017     sf::RenderWindow* window{};
00018     sf::VideoMode videoMode;
00019     sf::Event event{};
00020
00021     // scenes
00022     class BaseScene* scenes[constants::sceneVariables::SCENE_COUNT];
00023     constants::sceneVariables::Scene currentScene;
00024
00025     // buttons
00026     Button* submenuButton,
00027           *demoCodeButton;
00028
00029     void initWindow();
00030     void initScenes();
00031     void init();
00032
00033 public:
00034     Window();
00035     ~Window() = default;
00036
00037     const bool running() const;
00038
00039     void pollEvent();
00040     void update();
00041     void render();
00042 };
00043
00044 #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](#).

```
00003     {
00004         Window window;
00005
00006         while (window.running()) {
00007             window.update();
00008
00009             window.render();
00010         }
00011
00012         return 0;
00013 }
```

## 8.134 main.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Window.hpp"
00002
00003 int main() {
00004     Window window;
00005
00006     while (window.running()) {
00007         window.update();
00008
00009         window.render();
00010     }
00011
00012     return 0;
00013 }
```

## 8.135 README.md File Reference



# Index

- `_POSIX_C_SOURCE`
    - `FileDialog.h`, [275](#)
  - `~Array`
    - `Array`, [49](#)
  - `~ControlMenu`
    - `ControlMenu`, [84](#)
  - `~CustomTextbox`
    - `CustomTextbox`, [87](#)
  - `~EventAnimation`
    - `EventAnimation`, [115](#)
  - `~EventSquare`
    - `EventSquare`, [118](#)
  - `~MenuArray`
    - `MenuArray`, [144](#)
  - `~MenuDataStructure`
    - `MenuDataStructure`, [149](#)
  - `~MenuLinkedList`
    - `MenuLinkedList`, [153](#)
  - `~NodeInfo`
    - `NodeInfo`, [175](#)
  - `~SquareInfo`
    - `SquareInfo`, [229](#)
  - `~Vector`
    - `Vector< T >`, [250](#)
  - `~Window`
    - `Window`, [256](#)
  - `~executor`
    - `pfd::internal::executor`, [120](#)
- `abort`
  - `pfd`, [42](#)
- `abort_retry_ignore`
  - `pfd`, [43](#)
- `active`
  - `Square`, [225](#)
- `activeAddMode`
  - `MenuLinkedList`, [163](#)
- `activeCreateMode`
  - `MenuLinkedList`, [163](#)
- `activeOptionsMenu`
  - `MenuLinkedList`, [163](#)
- `activeUpdateMode`
  - `MenuLinkedList`, [164](#)
- `ADD_BUTTON`
  - `constants::MenuArray`, [17](#)
  - `constants::MenuLinkedList`, [32](#)
- `addModeEvents`
  - `CLLScene`, [74](#)
  - `DLLScene`, [94](#)
  - `DynamicArrayScene`, [106](#)
  - `SLLScene`, [216](#)
  - `StaticArrayScene`, [240](#)
- `addModeValue`
  - `MenuArray`, [147](#)
  - `MenuLinkedList`, [164](#)
- `addNode`
  - `LinkedList`, [133](#)
- `addSquare`
  - `Array`, [49](#)
- `addTextbox`
  - `MenuLinkedList`, [164](#)
- `ALLOCATE_BUTTON`
  - `constants::MenuArray`, [17](#)
- `allocateModeEvents`
  - `DynamicArrayScene`, [108](#)
- `allocateModeValue`
  - `MenuArray`, [147](#)
- `allocateSquare`
  - `Array`, [50](#)
- `Array`, [47](#)
  - `~Array`, [49](#)
  - `addSquare`, [49](#)
  - `allocateSquare`, [50](#)
  - `Array`, [48](#), [49](#)
  - `createArray`, [51](#)
  - `deleteSquare`, [51](#)
  - `DYNAMIC`, [48](#)
  - `findValue`, [52](#)
  - `getSize`, [52](#)
  - `getSquaresSize`, [52](#)
  - `init`, [52](#)
  - `initHighlighter`, [53](#)
  - `processControlMenu`, [53](#)
  - `render`, [53](#)
  - `renderHighlighter`, [54](#)
  - `resetEvents`, [54](#)
  - `searchSquare`, [54](#)
  - `setSpeed`, [54](#)
  - `STATIC`, [48](#)
  - `toggleLines`, [55](#)
  - `TypeArray`, [48](#)
  - `update`, [55](#)
  - `updateAnimation`, [55](#)
  - `updateSquare`, [56](#)
- `Arrow`, [56](#)
  - `Arrow`, [57](#)
  - `arrowSprite`, [61](#)
  - `arrowTexture`, [61](#)
  - `autoRotate`, [58](#)

- autoScale, 58
  - hasSetMid, 61
  - hide, 58
  - length, 61
  - points, 61
  - render, 59
  - resetColor, 59
  - setMid, 59
  - setPositions, 59
  - setStart, 60
  - show, 60
  - toggleActiveColor, 60
- arrowSprite
  - Arrow, 61
- arrowTexture
  - Arrow, 61
- ArrowType
  - NodeInfo, 174
- assign
  - Vector< T >, 250
- at
  - Vector< T >, 250
- autoRotate
  - Arrow, 58
  - BackArrow, 63
- autoScale
  - Arrow, 58
  - BackArrow, 63
- available
  - pfd::settings, 209
- back
  - Vector< T >, 251
- BackArrow, 62
  - autoRotate, 63
  - autoScale, 63
  - BackArrow, 62
  - hide, 64
  - render, 64
  - resetColor, 64
  - setPosition, 65
  - show, 65
  - toggleActiveColorNode, 65
- BaseDraw, 66
  - BaseDraw, 66
  - render, 66
  - window, 67
- BaseScene, 67
  - BaseScene, 68
  - controlMenu, 70
  - createModeButton, 68
  - isDemoCodeOpen, 70
  - isMenuOpen, 70
  - modeButton, 70
  - pollEvent, 69
  - render, 69
  - setWindow, 69
  - update, 69
  - window, 70
- begin
  - Vector< T >, 251
- Button, 71
  - Button, 71
  - checkClicked, 72
  - constants::ControlMenu, 14
  - constants::MenuArray, 17
  - constants::MenuArray::CreateMode, 21
  - constants::MenuDataStructure, 27
  - constants::MenuDataStructure::CreateMode, 28
  - constants::MenuLinkedList, 32
  - constants::MenuLinkedList::CreateMode, 35
  - getPosition, 72
  - getSize, 72
  - getTextString, 72
  - pollEvent, 72
  - render, 72
  - setColor, 73
  - update, 73
- button
  - pfd, 42
- BUTTON\_COUNT
  - constants::ControlMenu, 14
  - constants::MenuArray, 18
  - constants::MenuArray::CreateMode, 21
  - constants::MenuDataStructure, 27
  - constants::MenuDataStructure::CreateMode, 29
  - constants::MenuLinkedList, 32
  - constants::MenuLinkedList::CreateMode, 35
- BUTTON\_NAME\_SIZE
  - constants::ControlMenu, 14
  - constants::MenuArray, 18
  - constants::MenuDataStructure, 28
  - constants::MenuLinkedList, 32
- BUTTON\_NAMES
  - constants::ControlMenu, 14
  - constants::MenuArray, 18
  - constants::MenuArray::CreateMode, 22
  - constants::MenuDataStructure, 28
  - constants::MenuDataStructure::CreateMode, 29
  - constants::MenuLinkedList, 33
  - constants::MenuLinkedList::CreateMode, 35
- buttons
  - MenuLinkedList, 164
- buttons\_to\_name
  - pfd::internal::dialog, 90
- calculateEffectivePositions
  - LinkedList, 134
- cancel
  - pfd, 42
- check\_program
  - pfd::settings, 209
- checkClicked
  - Button, 72
- choice
  - pfd, 43
- chosen
  - Square, 225



- CIRCULAR
  - LinkedList, [132](#)
- CIRCULAR\_LINKED\_LIST\_SCENE
  - constants::sceneVariables, [40](#)
- clear
  - LinkedList, [134](#)
  - Vector< T >, [251](#)
- CLEAR\_BUTTON
  - constants::MenuDataStructure, [27](#)
- clickGray
  - constants, [11](#)
- clickGreen
  - constants, [12](#)
- CLLScene, [73](#)
  - addModeEvents, [74](#)
  - CLLScene, [74](#)
  - deleteModeEvents, [76](#)
  - pollEvent, [78](#)
  - render, [78](#)
  - reset, [79](#)
  - searchModeEvents, [79](#)
  - update, [80](#)
  - updateModeEvents, [81](#)
- CODES\_PATH
  - constants::Highlighter::DLL, [16](#)
  - constants::Highlighter::SLL, [16](#)
- codeScale
  - constants::Highlighter, [15](#)
- colorArrows
  - EventAnimation, [116](#)
- colorNodes
  - EventAnimation, [116](#)
- constants, [11](#)
  - clickGray, [11](#)
  - clickGreen, [12](#)
  - fontPath, [12](#)
  - hoverGray, [12](#)
  - hoverGreen, [12](#)
  - titleWindow, [13](#)
  - transparentGreen, [12](#)
- constants::Arrow, [13](#)
  - defaultScaleRectangle, [13](#)
  - sizeRectangle, [13](#)
- constants::ControlMenu, [13](#)
  - Button, [14](#)
  - BUTTON\_COUNT, [14](#)
  - BUTTON\_NAME\_SIZE, [14](#)
  - BUTTON\_NAMES, [14](#)
  - NEXT, [14](#)
  - None, [14](#)
  - PLAY, [14](#)
  - PREVIOUS, [14](#)
  - SPEED\_DOWN, [14](#)
  - SPEED\_UP, [14](#)
  - TEXT\_SIZE, [15](#)
- constants::Highlighter, [15](#)
  - codeScale, [15](#)
- constants::Highlighter::DLL, [15](#)
- CODES\_PATH, [16](#)
- constants::Highlighter::SLL, [16](#)
- CODES\_PATH, [16](#)
- constants::LinkedList, [16](#)
- constants::MenuArray, [16](#)
  - ADD\_BUTTON, [17](#)
  - ALLOCATE\_BUTTON, [17](#)
  - Button, [17](#)
  - BUTTON\_COUNT, [18](#)
  - BUTTON\_NAME\_SIZE, [18](#)
  - BUTTON\_NAMES, [18](#)
  - CREATE\_BUTTON, [17](#)
  - DELETE\_BUTTON, [17](#)
  - DYNAMIC, [17](#)
  - NONE, [17](#)
  - SEARCH\_BUTTON, [17](#)
  - STATIC, [17](#)
  - Type, [17](#)
  - UPDATE\_BUTTON, [17](#)
- constants::MenuArray::AddMode, [18](#)
  - NONE, [19](#)
  - POSITION\_TEXTBOX, [19](#)
  - Textbox, [18](#)
  - TEXTBOX\_COUNT, [19](#)
  - TEXTBOX\_LENGTH, [19](#)
  - TEXTBOX\_NAMES, [19](#)
  - VALUE\_TEXTBOX, [19](#)
- constants::MenuArray::AllocateMode, [20](#)
  - NONE, [20](#)
  - Textbox, [20](#)
  - TEXTBOX\_COUNT, [20](#)
  - TEXTBOX\_LENGTH, [20](#)
  - TEXTBOX\_NAME, [20](#)
  - VALUE\_TEXTBOX, [20](#)
- constants::MenuArray::CreateMode, [21](#)
  - Button, [21](#)
  - BUTTON\_COUNT, [21](#)
  - BUTTON\_NAMES, [22](#)
  - DEFINED\_LIST\_BUTTON, [21](#)
  - FILE\_BUTTON, [21](#)
  - NAME\_SIZE, [22](#)
  - NONE, [21](#)
  - RANDOM\_BUTTON, [21](#)
  - TEXTBOX\_COUNT, [22](#)
  - TEXTBOX\_LENGTH, [22](#)
  - TEXTBOX\_NAMES, [22](#)
- constants::MenuArray::DeleteMode, [23](#)
  - NONE, [23](#)
  - POSITION\_TEXTBOX, [23](#)
  - Textbox, [23](#)
  - TEXTBOX\_COUNT, [23](#)
  - TEXTBOX\_LENGTH, [24](#)
  - TEXTBOX\_NAME, [24](#)
- constants::MenuArray::SearchMode, [24](#)
  - NONE, [24](#)
  - Textbox, [24](#)
  - TEXTBOX\_COUNT, [25](#)
  - TEXTBOX\_LENGTH, [25](#)

- TEXTBOX\_NAME, 25
- VALUE\_TEXTBOX, 24
- constants::MenuArray::UpdateMode, 25
  - NONE, 26
  - POSITION\_TEXTBOX, 26
  - Textbox, 25
  - TEXTBOX\_COUNT, 26
  - TEXTBOX\_LENGTH, 26
  - TEXTBOX\_NAMES, 26
  - VALUE\_TEXTBOX, 26
- constants::MenuDataStructure, 27
  - Button, 27
  - BUTTON\_COUNT, 27
  - BUTTON\_NAME\_SIZE, 28
  - BUTTON\_NAMES, 28
  - CLEAR\_BUTTON, 27
  - CREATE\_BUTTON, 27
  - NONE, 27
  - POP\_BUTTON, 27
  - PUSH\_BUTTON, 27
- constants::MenuDataStructure::CreateMode, 28
  - Button, 28
  - BUTTON\_COUNT, 29
  - BUTTON\_NAMES, 29
  - DEFINED\_LIST\_BUTTON, 29
  - FILE\_BUTTON, 29
  - NAME\_SIZE, 29
  - NONE, 29
  - RANDOM\_BUTTON, 29
  - TEXTBOX\_COUNT, 29
  - TEXTBOX\_LENGTH, 30
  - TEXTBOX\_NAMES, 30
- constants::MenuDataStructure::PushMode, 30
  - NONE, 31
  - Textbox, 30
  - TEXTBOX\_COUNT, 31
  - TEXTBOX\_LENGTH, 31
  - TEXTBOX\_NAME, 31
  - VALUE\_TEXTBOX, 31
- constants::MenuLinkedList, 31
  - ADD\_BUTTON, 32
  - Button, 32
  - BUTTON\_COUNT, 32
  - BUTTON\_NAME\_SIZE, 32
  - BUTTON\_NAMES, 33
  - CREATE\_BUTTON, 32
  - DELETE\_BUTTON, 32
  - NONE, 32
  - SEARCH\_BUTTON, 32
  - UPDATE\_BUTTON, 32
- constants::MenuLinkedList::AddMode, 33
  - NONE, 33
  - POSITION\_TEXTBOX, 33
  - Textbox, 33
  - TEXTBOX\_COUNT, 34
  - TEXTBOX\_LENGTH, 34
  - TEXTBOX\_NAMES, 34
  - VALUE\_TEXTBOX, 33
- constants::MenuLinkedList::CreateMode, 34
  - Button, 35
  - BUTTON\_COUNT, 35
  - BUTTON\_NAMES, 35
  - DEFINED\_LIST\_BUTTON, 35
  - FILE\_BUTTON, 35
  - NAME\_SIZE, 35
  - NONE, 35
  - RANDOM\_BUTTON, 35
  - TEXTBOX\_COUNT, 36
  - TEXTBOX\_LENGTH, 36
  - TEXTBOX\_NAMES, 36
- constants::MenuLinkedList::DeleteMode, 36
  - NONE, 37
  - POSITION\_TEXTBOX, 37
  - Textbox, 36
  - TEXTBOX\_COUNT, 37
  - TEXTBOX\_LENGTH, 37
  - TEXTBOX\_NAME, 37
- constants::MenuLinkedList::SearchMode, 37
  - NONE, 38
  - Textbox, 37
  - TEXTBOX\_COUNT, 38
  - TEXTBOX\_LENGTH, 38
  - TEXTBOX\_NAME, 38
  - VALUE\_TEXTBOX, 38
- constants::MenuLinkedList::UpdateMode, 38
  - NONE, 39
  - POSITION\_TEXTBOX, 39
  - Textbox, 38
  - TEXTBOX\_COUNT, 39
  - TEXTBOX\_LENGTH, 39
  - TEXTBOX\_NAMES, 39
  - VALUE\_TEXTBOX, 39
- constants::NodeInfo, 40
- constants::sceneVariables, 40
  - CIRCULAR\_LINKED\_LIST\_SCENE, 40
  - DOUBLY\_LINKED\_LIST\_SCENE, 40
  - DYNAMIC\_ARRAY\_SCENE, 40
  - MAIN\_MENU\_SCENE, 40
  - NAME\_MODE\_BUTTON, 40
  - QUEUE\_SCENE, 40
  - Scene, 40
  - SCENE\_COUNT, 41
  - SCENE\_NAMES, 41
  - SINGLY\_LINKED\_LIST\_SCENE, 40
  - STACK\_SCENE, 40
  - STATIC\_ARRAY\_SCENE, 40
- constants::Square, 41
- constants::TitleNode, 41
- ControlMenu, 82
  - ~ControlMenu, 84
  - ControlMenu, 84
  - getSpeed, 84
  - getStatus, 84
  - NEXT, 83
  - None, 83
  - PAUSE, 83

- PLAY, 83
- pollEvents, 85
- PREVIOUS, 83
- render, 85
- reset, 85
- status, 86
- StatusCode, 83
- update, 86
- controlMenu
  - BaseScene, 70
- CREATE\_BUTTON
  - constants::MenuArray, 17
  - constants::MenuDataStructure, 27
  - constants::MenuLinkedList, 32
- createArray
  - Array, 51
- createLinkedList
  - LinkedList, 135
- createModeButton
  - BaseScene, 68
- createModeValue
  - MenuArray, 147
  - MenuDataStructure, 151
  - MenuLinkedList, 164
- createTextbox
  - MenuLinkedList, 164
- CustomTextbox, 86
  - ~CustomTextbox, 87
  - CustomTextbox, 87
  - getTextString, 88
  - pollEvent, 88
  - render, 88
  - resetInput, 88
  - update, 88
- data
  - Vector< T >, 251
- defaultScaleRectangle
  - constants::Arrow, 13
- DEFINED\_LIST\_BUTTON
  - constants::MenuArray::CreateMode, 21
  - constants::MenuDataStructure::CreateMode, 29
  - constants::MenuLinkedList::CreateMode, 35
- DELETE\_BUTTON
  - constants::MenuArray, 17
  - constants::MenuLinkedList, 32
- deleteModeEvents
  - CLLScene, 76
  - DLLScene, 98
  - DynamicArrayScene, 109
  - SLLScene, 218
  - StaticArrayScene, 241
- deleteModeValue
  - MenuArray, 147
  - MenuLinkedList, 165
- deleteNode
  - LinkedList, 136
- deleteSquare
  - Array, 51
- deleteTextbox
  - MenuLinkedList, 165
- desktop\_helper
  - pfd::internal::dialog, 91
- destroyArrow
  - NodeInfo, 176
- dialog
  - pfd::internal::dialog, 90
  - pfd::internal::executor, 122
- DLLScene, 93
  - addModeEvents, 94
  - deleteModeEvents, 98
  - DLLScene, 94
  - pollEvent, 100
  - render, 101
  - reset, 101
  - searchModeEvents, 101
  - update, 102
  - updateModeEvents, 104
- DOUBLY
  - LinkedList, 132
- DOUBLY\_LINKED\_LIST\_SCENE
  - constants::sceneVariables, 40
- DYNAMIC
  - Array, 48
  - constants::MenuArray, 17
- DYNAMIC\_ARRAY\_SCENE
  - constants::sceneVariables, 40
- DynamicArrayScene, 105
  - addModeEvents, 106
  - allocateModeEvents, 108
  - deleteModeEvents, 109
  - DynamicArrayScene, 106
  - pollEvent, 110
  - render, 110
  - reset, 111
  - searchModeEvents, 111
  - update, 111
  - updateModeEvents, 113
- Effective
  - NodeInfo, 175
- empty
  - Vector< T >, 251
- end
  - Vector< T >, 252
- erase
  - Vector< T >, 252
- error
  - pfd, 43
- EventAnimation, 114
  - ~EventAnimation, 115
  - colorArrows, 116
  - colorNodes, 116
  - EventAnimation, 114
  - eventSquares, 116
  - eventSquaresTemp, 116
  - hiddenArrows, 116
  - indexBackArrow, 116

- isPrintNormal, 117
- isPrintPreVal, 117
- isShowBackArrow, 117
- lines, 117
- operator=, 115
- reset, 115
- statusChosenNode, 117
- titleNodes, 117
- EventSquare, 118
  - ~EventSquare, 118
  - EventSquare, 118
  - isPrintPreVal, 119
  - status, 119
  - title, 119
- eventSquares
  - EventAnimation, 116
- eventSquaresTemp
  - EventAnimation, 116
- FILE\_BUTTON
  - constants::MenuArray::CreateMode, 21
  - constants::MenuDataStructure::CreateMode, 29
  - constants::MenuLinkedList::CreateMode, 35
- file\_dialog
  - pfd::internal::file\_dialog, 125
- FileDialog.h
  - \_POSIX\_C\_SOURCE, 275
  - PFD\_HAS\_IFILEDIALOG, 275
  - PFD\_OSX\_ICON, 276
- findValue
  - Array, 52
  - LinkedList, 136
- flag
  - pfd::settings, 208
- flags
  - pfd::settings, 210
- folder
  - pfd::internal::file\_dialog, 125
- font
  - SingleNode, 214
  - Square, 227
- fontPath
  - constants, 12
- force\_overwrite
  - pfd, 44
- force\_path
  - pfd, 44
- front
  - Vector< T >, 252
- get\_icon\_name
  - pfd::internal::dialog, 91
- getActiveCreateMode
  - MenuArray, 144
  - MenuDataStructure, 149
  - MenuLinkedList, 153
- getActiveOptionsMenu
  - MenuArray, 144
  - MenuDataStructure, 149
- MenuLinkedList, 154
- getBox
  - TextBox, 247
- getButton
  - MenuArray, 144
  - MenuDataStructure, 149
  - MenuLinkedList, 154
- getCapacity
  - Vector< T >, 253
- getCornersRadius
  - sf::RoundedRectangleShape, 199
- getPoint
  - sf::RoundedRectangleShape, 199
- getPointCount
  - sf::RoundedRectangleShape, 200
- getPosition
  - Button, 72
  - NodeInfo, 176
  - SingleNode, 213
  - Square, 226
- getPosNode
  - LinkedList, 136
- getSize
  - Array, 52
  - Button, 72
  - LinkedList, 137
  - sf::RoundedRectangleShape, 200
- getSpeed
  - ControlMenu, 84
- getSquaresSize
  - Array, 52
- getStatus
  - ControlMenu, 84
  - Square, 226
  - SquareInfo, 229
- getStatusNode
  - NodeInfo, 176
- getTextString
  - Button, 72
  - CustomTextbox, 88
  - TextBox, 247
- getValue
  - NodeInfo, 176
  - SquareInfo, 229
- has\_kdialog
  - pfd::settings, 208
- has\_matedialog
  - pfd::settings, 208
- has\_qarma
  - pfd::settings, 208
- has\_zenity
  - pfd::settings, 208
- hasSetMid
  - Arrow, 61
- hidden
  - Square, 225
- hiddenArrows
  - EventAnimation, 116

- hide
  - Arrow, 58
  - BackArrow, 64
  - NodeInfo, 177
- Highlighter, 129
  - Highlighter, 130
  - render, 130
  - resetToggle, 130
  - toggle, 131
- home
  - pfd::path, 188
- hoverGray
  - constants, 12
- hoverGreen
  - constants, 12
- icon
  - pfd, 43
- ignore
  - pfd, 42
- inactive
  - Square, 225
- InChain
  - NodeInfo, 174
- include/Constants.hpp, 259, 262
- include/core/Array.cpp, 267
- include/core/Array.hpp, 271, 272
- include/core/EventAnimation.cpp, 272, 273
- include/core/EventAnimation.hpp, 273
- include/core/FileDialog.h, 274, 276
- include/core/LinkedList.cpp, 298
- include/core/LinkedList.hpp, 303
- include/core/Random.h, 304, 305
- include/core/Vector.h, 305
- include/draw/Arrow.cpp, 308
- include/draw/Arrow.hpp, 310
- include/draw/BackArrow.cpp, 310, 311
- include/draw/BackArrow.hpp, 312
- include/draw/BaseDraw.cpp, 313
- include/draw/BaseDraw.hpp, 313, 314
- include/draw/NodeInfo.cpp, 314
- include/draw/NodeInfo.hpp, 317
- include/draw/SingleNode.cpp, 318
- include/draw/SingleNode.hpp, 319
- include/draw/Square.cpp, 320
- include/draw/Square.hpp, 321
- include/draw/SquareInfo.cpp, 322
- include/draw/SquareInfo.hpp, 323
- include/libScene/AllScenes.hpp, 324
- include/libScene/BaseScene.cpp, 324
- include/libScene/BaseScene.hpp, 325
- include/libScene/CLLScene.cpp, 325, 326
- include/libScene/CLLScene.hpp, 332, 333
- include/libScene/ControlMenu.cpp, 333
- include/libScene/ControlMenu.hpp, 334, 335
- include/libScene/DLLScene.cpp, 335, 336
- include/libScene/DLLScene.hpp, 345
- include/libScene/DynamicArrayScene.cpp, 346
- include/libScene/DynamicArrayScene.hpp, 352
- include/libScene/Highlighter.cpp, 352, 353
- include/libScene/Highlighter.hpp, 353, 354
- include/libScene/MainMenu.cpp, 354
- include/libScene/MainMenu.hpp, 354, 355
- include/libScene/MenuArray.cpp, 355
- include/libScene/MenuArray.hpp, 361
- include/libScene/MenuDataStructure.cpp, 362
- include/libScene/MenuDataStructure.hpp, 365, 366
- include/libScene/MenuLinkedList.cpp, 367
- include/libScene/MenuLinkedList.hpp, 372
- include/libScene/QueueScene.cpp, 373
- include/libScene/QueueScene.hpp, 378
- include/libScene/SLLScene.cpp, 379
- include/libScene/SLLScene.hpp, 385, 386
- include/libScene/StackScene.cpp, 386
- include/libScene/StackScene.hpp, 391
- include/libScene/StaticArrayScene.cpp, 392
- include/libScene/StaticArrayScene.hpp, 396
- include/MousePosition.cpp, 396, 397
- include/MousePosition.hpp, 397
- include/stuff/button.cpp, 397, 398
- include/stuff/button.hpp, 399
- include/stuff/CustomTextbox.cpp, 400
- include/stuff/CustomTextbox.hpp, 401
- include/stuff/RoundedRectangleShape.cpp, 402
- include/stuff/RoundedRectangleShape.hpp, 403, 404
- include/stuff/Textbox.cpp, 404
- include/stuff/Textbox.hpp, 406
- include/stuff/ToStringWithPrecision.hpp, 407
- include/Window.cpp, 407, 408
- include/Window.hpp, 410
- indexBackArrow
  - EventAnimation, 116
- info
  - pfd, 43
- init
  - Array, 52
  - MenuLinkedList, 154
- initAddMode
  - MenuLinkedList, 154
- initArrow
  - NodeInfo, 177
- initButtons
  - MenuLinkedList, 155
- initCreateMode
  - MenuLinkedList, 155
- initDeleteMode
  - MenuLinkedList, 156
- initHighlighter
  - Array, 53
  - LinkedList, 137
- initSearchMode
  - MenuLinkedList, 156
- initUpdateMode
  - MenuLinkedList, 156
- insert
  - Vector< T >, 253
- is\_kdialog

- pfd::settings, 210
- is\_osascript
  - pfd::settings, 210
- is\_scanned
  - pfd::settings, 208
- is\_verbose
  - pfd::settings, 208
- is\_vista
  - pfd::settings, 208
- is\_zenity
  - pfd::settings, 211
- isDemoCodeOpen
  - BaseScene, 70
- isMenuOpen
  - BaseScene, 70
- isOpenFileDialog
  - MenuLinkedList, 165
- isPrintNormal
  - EventAnimation, 117
- isPrintPreVal
  - EventAnimation, 117
  - EventSquare, 119
- isShowBackArrow
  - EventAnimation, 117
- kill
  - pfd::internal::dialog, 91
  - pfd::internal::executor, 120
- LEFT
  - NodeInfo, 174
- length
  - Arrow, 61
- lines
  - EventAnimation, 117
- LinkedList, 131
  - addNode, 133
  - calculateEffectivePositions, 134
  - CIRCULAR, 132
  - clear, 134
  - createLinkedList, 135
  - deleteNode, 136
  - DOUBLY, 132
  - findValue, 136
  - getPosNode, 136
  - getSize, 137
  - initHighlighter, 137
  - LinkedList, 132, 133
  - processControlMenu, 137
  - render, 137
  - renderHighlighter, 138
  - resetEvents, 138
  - searchNode, 138
  - setSpeed, 139
  - SINGLY, 132
  - toggleLines, 139
  - TypeLinkedList, 132
  - update, 139
  - updateAnimation, 139
  - updateNode, 140
- m\_async
  - pfd::internal::dialog, 93
- main
  - main.cpp, 411
- main.cpp, 410
  - main, 411
- MAIN\_MENU\_SCENE
  - constants::sceneVariables, 40
- MainMenu, 141
  - MainMenu, 142
  - pollEvent, 142
  - render, 142
  - update, 143
- max\_flag
  - pfd::settings, 208
- MenuArray, 143
  - ~MenuArray, 144
  - addModeValue, 147
  - allocateModeValue, 147
  - createModeValue, 147
  - deleteModeValue, 147
  - getActiveCreateMode, 144
  - getActiveOptionsMenu, 144
  - getButton, 144
  - MenuArray, 144
  - pollEvents, 145
  - render, 145
  - resetActiveOptionsMenu, 146
  - searchModeValue, 147
  - update, 146
  - updateModeValue, 147
- MenuDataStructure, 148
  - ~MenuDataStructure, 149
  - createModeValue, 151
  - getActiveCreateMode, 149
  - getActiveOptionsMenu, 149
  - getButton, 149
  - MenuDataStructure, 148
  - pollEvents, 149
  - pushModeValue, 151
  - render, 150
  - resetActiveOptionsMenu, 150
  - resetActiveOptionsMenuOnly, 150
  - update, 150
- MenuLinkedList, 151
  - ~MenuLinkedList, 153
  - activeAddMode, 163
  - activeCreateMode, 163
  - activeOptionsMenu, 163
  - activeUpdateMode, 164
  - addModeValue, 164
  - addTextbox, 164
  - buttons, 164
  - createModeValue, 164
  - createTextbox, 164
  - deleteModeValue, 165
  - deleteTextbox, 165

- getActiveCreateMode, [153](#)
- getActiveOptionsMenu, [154](#)
- getButton, [154](#)
- init, [154](#)
- initAddMode, [154](#)
- initButtons, [155](#)
- initCreateMode, [155](#)
- initDeleteMode, [156](#)
- initSearchMode, [156](#)
- initUpdateMode, [156](#)
- isOpenFileDialog, [165](#)
- MenuLinkedList, [153](#)
- pollEventAddMode, [157](#)
- pollEventCreateMode, [157](#)
- pollEventDeleteMode, [157](#)
- pollEvents, [158](#)
- pollEventSearchMode, [158](#)
- pollEventUpdateMode, [158](#)
- render, [159](#)
- renderAddMode, [159](#)
- renderCreateMode, [159](#)
- renderDeleteMode, [160](#)
- renderSearchMode, [160](#)
- renderUpdateMode, [160](#)
- resetActiveOptionsMenu, [160](#)
- searchModeValue, [165](#)
- searchTextbox, [165](#)
- subCreateMode, [165](#)
- update, [160](#)
- updateAddMode, [161](#)
- updateCreateMode, [161](#)
- updateDeleteMode, [162](#)
- updateModeValue, [166](#)
- updateSearchMode, [162](#)
- updateTextbox, [166](#)
- updateUpdateMode, [163](#)
- window, [166](#)
- message
  - pfd::message, [168](#)
- modeButton
  - BaseScene, [70](#)
- mousePos
  - MousePosition, [172](#)
- MousePosition, [171](#)
  - mousePos, [172](#)
  - mousePosView, [172](#)
  - relativeWindow, [172](#)
  - updateMousePosition, [172](#)
- mousePosView
  - MousePosition, [172](#)
- multiselect
  - pfd, [44](#)
- NAME\_MODE\_BUTTON
  - constants::sceneVariables, [40](#)
- NAME\_SIZE
  - constants::MenuArray::CreateMode, [22](#)
  - constants::MenuDataStructure::CreateMode, [29](#)
  - constants::MenuLinkedList::CreateMode, [35](#)
- NEXT
  - constants::ControlMenu, [14](#)
  - ControlMenu, [83](#)
- no
  - pfd, [42](#)
- NodeInfo, [173](#)
  - ~NodeInfo, [175](#)
  - ArrowType, [174](#)
  - destroyArrow, [176](#)
  - Effective, [175](#)
  - getPosition, [176](#)
  - getStatusNode, [176](#)
  - getValue, [176](#)
  - hide, [177](#)
  - InChain, [174](#)
  - initArrow, [177](#)
  - LEFT, [174](#)
  - NodeInfo, [175](#)
  - Normal, [175](#)
  - OutChain, [174](#)
  - Outside, [175](#)
  - reInitPos, [177](#)
  - reInitPreVal, [177](#)
  - render, [178](#)
  - reset, [178](#)
  - resetColorArrow, [178](#)
  - resetColorNode, [178](#)
  - resetTitle, [179](#)
  - RIGHT, [174](#)
  - setArrows, [179](#)
  - setEffectivePosition, [179](#)
  - setNodeInChain, [179](#)
  - setNodeOutside, [179](#)
  - setNodeVisible, [180](#)
  - setPrintNormal, [180](#)
  - setPrintPreVal, [180](#)
  - setTitle, [180](#)
  - setValue, [180](#)
  - show, [181](#)
  - StatusNode, [174](#)
  - toggleActiveColorArrow, [181](#)
  - toggleActiveColorNode, [181](#)
  - TypeNode, [175](#)
  - updateArrows, [181](#)
  - updateNode, [182](#)
  - Visible, [174](#)
- NONE
  - constants::MenuArray, [17](#)
  - constants::MenuArray::AddMode, [19](#)
  - constants::MenuArray::AllocateMode, [20](#)
  - constants::MenuArray::CreateMode, [21](#)
  - constants::MenuArray::DeleteMode, [23](#)
  - constants::MenuArray::SearchMode, [24](#)
  - constants::MenuArray::UpdateMode, [26](#)
  - constants::MenuDataStructure, [27](#)
  - constants::MenuDataStructure::CreateMode, [29](#)
  - constants::MenuDataStructure::PushMode, [31](#)
  - constants::MenuLinkedList, [32](#)

- constants::MenuLinkedList::AddMode, 33
- constants::MenuLinkedList::CreateMode, 35
- constants::MenuLinkedList::DeleteMode, 37
- constants::MenuLinkedList::SearchMode, 38
- constants::MenuLinkedList::UpdateMode, 39
- None
  - constants::ControlMenu, 14
  - ControlMenu, 83
- none
  - pfd, 44
- Normal
  - NodeInfo, 175
- notify
  - pfd::notify, 184
- ok
  - pfd, 42, 43
- ok\_cancel
  - pfd, 43
- open
  - pfd::internal::file\_dialog, 125
- open\_file
  - pfd::open\_file, 187
- operator<<
  - pfd, 44
- operator=
  - EventAnimation, 115
  - Vector< T >, 253
- operator&
  - pfd, 44
- operator[]
  - Vector< T >, 253
- operator |
  - pfd, 44
- opt
  - pfd, 43
- osascript\_quote
  - pfd::internal::dialog, 92
- OutChain
  - NodeInfo, 174
- Outside
  - NodeInfo, 175
- PAUSE
  - ControlMenu, 83
- pfd, 41
  - abort, 42
  - abort\_retry\_ignore, 43
  - button, 42
  - cancel, 42
  - choice, 43
  - error, 43
  - force\_overwrite, 44
  - force\_path, 44
  - icon, 43
  - ignore, 42
  - info, 43
  - multiselect, 44
  - no, 42
  - none, 44
  - ok, 42, 43
  - ok\_cancel, 43
  - operator<<, 44
  - operator&, 44
  - operator |, 44
  - opt, 43
  - question, 43
  - retry, 42
  - retry\_cancel, 43
  - warning, 43
  - yes, 42
  - yes\_no, 43
  - yes\_no\_cancel, 43
- pfd::internal, 45
- pfd::internal::dialog, 89
  - buttons\_to\_name, 90
  - desktop\_helper, 91
  - dialog, 90
  - get\_icon\_name, 91
  - kill, 91
  - m\_async, 93
  - osascript\_quote, 92
  - powershell\_quote, 92
  - ready, 92
  - shell\_quote, 92
- pfd::internal::executor, 119
  - ~executor, 120
  - dialog, 122
  - kill, 120
  - ready, 120
  - result, 121
  - start\_process, 121
  - stop, 122
- pfd::internal::file\_dialog, 123
  - file\_dialog, 125
  - folder, 125
  - open, 125
  - save, 125
  - string\_result, 129
  - type, 124
  - vector\_result, 129
- pfd::internal::platform, 190
- pfd::message, 166
  - message, 168
  - result, 170
- pfd::notify, 182
  - notify, 184
- pfd::open\_file, 185
  - open\_file, 187
  - result, 188
- pfd::path, 188
  - home, 188
  - separator, 189
- pfd::save\_file, 202
  - result, 204
  - save\_file, 204
- pfd::select\_folder, 205



- result, [207](#)
- select\_folder, [206](#)
- pfd::settings, [207](#)
  - available, [209](#)
  - check\_program, [209](#)
  - flag, [208](#)
  - flags, [210](#)
  - has\_kdialog, [208](#)
  - has\_matedialog, [208](#)
  - has\_qarma, [208](#)
  - has\_zenity, [208](#)
  - is\_kdialog, [210](#)
  - is\_osascript, [210](#)
  - is\_scanned, [208](#)
  - is\_verbose, [208](#)
  - is\_vista, [208](#)
  - is\_zenity, [211](#)
  - max\_flag, [208](#)
  - rescan, [211](#)
  - settings, [208](#)
  - verbose, [211](#)
- PFD\_HAS\_IFILEDIALOG
  - FileDialog.h, [275](#)
- PFD\_OSX\_ICON
  - FileDialog.h, [276](#)
- PLAY
  - constants::ControlMenu, [14](#)
  - ControlMenu, [83](#)
- points
  - Arrow, [61](#)
- pollEvent
  - BaseScene, [69](#)
  - Button, [72](#)
  - CLLScene, [78](#)
  - CustomTextbox, [88](#)
  - DLLScene, [100](#)
  - DynamicArrayScene, [110](#)
  - MainMenu, [142](#)
  - QueueScene, [191](#)
  - SLLScene, [220](#)
  - StackScene, [233](#)
  - StaticArrayScene, [242](#)
  - TextBox, [247](#)
  - Window, [256](#)
- pollEventAddMode
  - MenuLinkedList, [157](#)
- pollEventCreateMode
  - MenuLinkedList, [157](#)
- pollEventDeleteMode
  - MenuLinkedList, [157](#)
- pollEvents
  - ControlMenu, [85](#)
  - MenuArray, [145](#)
  - MenuDataStructure, [149](#)
  - MenuLinkedList, [158](#)
- pollEventSearchMode
  - MenuLinkedList, [158](#)
- pollEventUpdateMode
  - MenuLinkedList, [158](#)
- pop\_back
  - Vector< T >, [254](#)
- POP\_BUTTON
  - constants::MenuDataStructure, [27](#)
- popModeEvents
  - QueueScene, [192](#)
  - StackScene, [233](#)
- POSITION\_TEXTBOX
  - constants::MenuArray::AddMode, [19](#)
  - constants::MenuArray::DeleteMode, [23](#)
  - constants::MenuArray::UpdateMode, [26](#)
  - constants::MenuLinkedList::AddMode, [33](#)
  - constants::MenuLinkedList::DeleteMode, [37](#)
  - constants::MenuLinkedList::UpdateMode, [39](#)
- powershell\_quote
  - pfd::internal::dialog, [92](#)
- PREVIOUS
  - constants::ControlMenu, [14](#)
  - ControlMenu, [83](#)
- processControlMenu
  - Array, [53](#)
  - LinkedList, [137](#)
- push\_back
  - Vector< T >, [254](#)
- PUSH\_BUTTON
  - constants::MenuDataStructure, [27](#)
- pushModeEvents
  - QueueScene, [194](#)
  - StackScene, [235](#)
- pushModeValue
  - MenuDataStructure, [151](#)
- question
  - pfd, [43](#)
- QUEUE\_SCENE
  - constants::sceneVariables, [40](#)
- QueueScene, [190](#)
  - pollEvent, [191](#)
  - popModeEvents, [192](#)
  - pushModeEvents, [194](#)
  - QueueScene, [191](#)
  - render, [195](#)
  - reset, [195](#)
  - update, [196](#)
- Random, [45](#)
- RANDOM\_BUTTON
  - constants::MenuArray::CreateMode, [21](#)
  - constants::MenuDataStructure::CreateMode, [29](#)
  - constants::MenuLinkedList::CreateMode, [35](#)
- README.md, [411](#)
- ready
  - pfd::internal::dialog, [92](#)
  - pfd::internal::executor, [120](#)
- reInitPos
  - NodeInfo, [177](#)
- reInitPreVal
  - NodeInfo, [177](#)

- relativeWindow
  - MousePosition, 172
- render
  - Array, 53
  - Arrow, 59
  - BackArrow, 64
  - BaseDraw, 66
  - BaseScene, 69
  - Button, 72
  - CLLScene, 78
  - ControlMenu, 85
  - CustomTextbox, 88
  - DLLScene, 101
  - DynamicArrayScene, 110
  - Highlighter, 130
  - LinkedList, 137
  - MainMenu, 142
  - MenuArray, 145
  - MenuDataStructure, 150
  - MenuLinkedList, 159
  - NodeInfo, 178
  - QueueScene, 195
  - SingleNode, 213
  - SLLScene, 220
  - Square, 226
  - SquareInfo, 229
  - StackScene, 237
  - StaticArrayScene, 242
  - TextBox, 247
  - Window, 257
- renderAddMode
  - MenuLinkedList, 159
- renderCreateMode
  - MenuLinkedList, 159
- renderDeleteMode
  - MenuLinkedList, 160
- renderHighlighter
  - Array, 54
  - LinkedList, 138
- renderSearchMode
  - MenuLinkedList, 160
- renderUpdateMode
  - MenuLinkedList, 160
- rescan
  - pfd::settings, 211
- reset
  - CLLScene, 79
  - ControlMenu, 85
  - DLLScene, 101
  - DynamicArrayScene, 111
  - EventAnimation, 115
  - NodeInfo, 178
  - QueueScene, 195
  - SLLScene, 220
  - SquareInfo, 230
  - StackScene, 237
  - StaticArrayScene, 243
- resetActiveOptionsMenu
  - MenuArray, 146
  - MenuDataStructure, 150
  - MenuLinkedList, 160
- resetActiveOptionsMenuOnly
  - MenuDataStructure, 150
- resetColor
  - Arrow, 59
  - BackArrow, 64
  - SingleNode, 213
  - Square, 226
- resetColorArrow
  - NodeInfo, 178
- resetColorNode
  - NodeInfo, 178
- resetEvents
  - Array, 54
  - LinkedList, 138
- resetInput
  - CustomTextbox, 88
  - TextBox, 248
- resetTitle
  - NodeInfo, 179
  - SquareInfo, 230
- resetToggle
  - Highlighter, 130
- resize
  - Vector< T >, 254
- result
  - pfd::internal::executor, 121
  - pfd::message, 170
  - pfd::open\_file, 188
  - pfd::save\_file, 204
  - pfd::select\_folder, 207
- retry
  - pfd, 42
- retry\_cancel
  - pfd, 43
- RIGHT
  - NodeInfo, 174
- RoundedRectangleShape
  - sf::RoundedRectangleShape, 198
- running
  - Window, 257
- save
  - pfd::internal::file\_dialog, 125
- save\_file
  - pfd::save\_file, 204
- Scene
  - constants::sceneVariables, 40
- SCENE\_COUNT
  - constants::sceneVariables, 41
- SCENE\_NAMES
  - constants::sceneVariables, 41
- SEARCH\_BUTTON
  - constants::MenuArray, 17
  - constants::MenuLinkedList, 32
- searchModeEvents
  - CLLScene, 79

- DLLScene, 101
- DynamicArrayScene, 111
- SLLScene, 220
- StaticArrayScene, 243
- searchModeValue
  - MenuArray, 147
  - MenuLinkedList, 165
- searchNode
  - LinkedList, 138
- searchSquare
  - Array, 54
- searchTextbox
  - MenuLinkedList, 165
- select\_folder
  - pfd::select\_folder, 206
- separator
  - pfd::path, 189
- setArrows
  - NodeInfo, 179
- setColor
  - Button, 73
- setCornerPointCount
  - sf::RoundedRectangleShape, 200
- setCornersRadius
  - sf::RoundedRectangleShape, 201
- setEffectivePosition
  - NodeInfo, 179
- setMid
  - Arrow, 59
- setNodeInChain
  - NodeInfo, 179
- setNodeOutside
  - NodeInfo, 179
- setNodeVisible
  - NodeInfo, 180
- setPosition
  - BackArrow, 65
  - SingleNode, 214
  - Square, 226
- setPositions
  - Arrow, 59
- setPrintNormal
  - NodeInfo, 180
- setPrintPreVal
  - NodeInfo, 180
  - SquareInfo, 230
- setSize
  - sf::RoundedRectangleShape, 201
- setSpeed
  - Array, 54
  - LinkedList, 139
- setStart
  - Arrow, 60
- setStatus
  - Square, 227
  - SquareInfo, 230
- setText
  - SingleNode, 214
  - Square, 227
- settings
  - pfd::settings, 208
- setTitle
  - NodeInfo, 180
  - SquareInfo, 231
- setValue
  - NodeInfo, 180
  - SquareInfo, 231
- setWindow
  - BaseScene, 69
- sf, 45
- sf::RoundedRectangleShape, 197
  - getCornersRadius, 199
  - getPoint, 199
  - getPointCount, 200
  - getSize, 200
  - RoundedRectangleShape, 198
  - setCornerPointCount, 200
  - setCornersRadius, 201
  - setSize, 201
- shell\_quote
  - pfd::internal::dialog, 92
- show
  - Arrow, 60
  - BackArrow, 65
  - NodeInfo, 181
- SingleNode, 212
  - font, 214
  - getPosition, 213
  - render, 213
  - resetColor, 213
  - setPosition, 214
  - setText, 214
  - SingleNode, 212
  - toggleActiveColor, 214
- SINGLY
  - LinkedList, 132
- SINGLY\_LINKED\_LIST\_SCENE
  - constants::sceneVariables, 40
- size
  - Vector< T >, 254
- sizeRectangle
  - constants::Arrow, 13
- SLLScene, 215
  - addModeEvents, 216
  - deleteModeEvents, 218
  - pollEvent, 220
  - render, 220
  - reset, 220
  - searchModeEvents, 220
  - SLLScene, 216
  - update, 221
  - updateModeEvents, 223
- SPEED\_DOWN
  - constants::ControlMenu, 14
- SPEED\_UP
  - constants::ControlMenu, 14

- Square, 224
  - active, 225
  - chosen, 225
  - font, 227
  - getPosition, 226
  - getStatus, 226
  - hidden, 225
  - inactive, 225
  - render, 226
  - resetColor, 226
  - setPosition, 226
  - setStatus, 227
  - setText, 227
  - Square, 225
  - Status, 225
- SquareInfo, 228
  - ~SquareInfo, 229
  - getStatus, 229
  - getValue, 229
  - render, 229
  - reset, 230
  - resetTitle, 230
  - setPrintPreVal, 230
  - setStatus, 230
  - setTitle, 231
  - setValue, 231
  - SquareInfo, 228
  - update, 231
- STACK\_SCENE
  - constants::sceneVariables, 40
- StackScene, 232
  - pollEvent, 233
  - popModeEvents, 233
  - pushModeEvents, 235
  - render, 237
  - reset, 237
  - StackScene, 233
  - update, 237
- start\_process
  - pfd::internal::executor, 121
- STATIC
  - Array, 48
  - constants::MenuArray, 17
- STATIC\_ARRAY\_SCENE
  - constants::sceneVariables, 40
- StaticArrayScene, 239
  - addModeEvents, 240
  - deleteModeEvents, 241
  - pollEvent, 242
  - render, 242
  - reset, 243
  - searchModeEvents, 243
  - StaticArrayScene, 240
  - update, 243
  - updateModeEvents, 245
- Status
  - Square, 225
- status
  - ControlMenu, 86
  - EventSquare, 119
  - statusChosenNode
    - EventAnimation, 117
  - StatusCode
    - ControlMenu, 83
  - StatusNode
    - NodeInfo, 174
  - stop
    - pfd::internal::executor, 122
  - string\_result
    - pfd::internal::file\_dialog, 129
  - subCreateMode
    - MenuLinkedList, 165
- TEXT\_SIZE
  - constants::ControlMenu, 15
- TextBox, 246
  - getBox, 247
  - getTextString, 247
  - pollEvent, 247
  - render, 247
  - resetInput, 248
  - TextBox, 246
  - update, 248
- Textbox
  - constants::MenuArray::AddMode, 18
  - constants::MenuArray::AllocateMode, 20
  - constants::MenuArray::DeleteMode, 23
  - constants::MenuArray::SearchMode, 24
  - constants::MenuArray::UpdateMode, 25
  - constants::MenuDataStructure::PushMode, 30
  - constants::MenuLinkedList::AddMode, 33
  - constants::MenuLinkedList::DeleteMode, 36
  - constants::MenuLinkedList::SearchMode, 37
  - constants::MenuLinkedList::UpdateMode, 38
- TEXTBOX\_COUNT
  - constants::MenuArray::AddMode, 19
  - constants::MenuArray::AllocateMode, 20
  - constants::MenuArray::CreateMode, 22
  - constants::MenuArray::DeleteMode, 23
  - constants::MenuArray::SearchMode, 25
  - constants::MenuArray::UpdateMode, 26
  - constants::MenuDataStructure::CreateMode, 29
  - constants::MenuDataStructure::PushMode, 31
  - constants::MenuLinkedList::AddMode, 34
  - constants::MenuLinkedList::CreateMode, 36
  - constants::MenuLinkedList::DeleteMode, 37
  - constants::MenuLinkedList::SearchMode, 38
  - constants::MenuLinkedList::UpdateMode, 39
- TEXTBOX\_LENGTH
  - constants::MenuArray::AddMode, 19
  - constants::MenuArray::AllocateMode, 20
  - constants::MenuArray::CreateMode, 22
  - constants::MenuArray::DeleteMode, 24
  - constants::MenuArray::SearchMode, 25
  - constants::MenuArray::UpdateMode, 26
  - constants::MenuDataStructure::CreateMode, 30
  - constants::MenuDataStructure::PushMode, 31

- constants::MenuLinkedList::AddMode, 34
- constants::MenuLinkedList::CreateMode, 36
- constants::MenuLinkedList::DeleteMode, 37
- constants::MenuLinkedList::SearchMode, 38
- constants::MenuLinkedList::UpdateMode, 39
- TEXTBOX\_NAME
  - constants::MenuArray::AllocateMode, 20
  - constants::MenuArray::DeleteMode, 24
  - constants::MenuArray::SearchMode, 25
  - constants::MenuDataStructure::PushMode, 31
  - constants::MenuLinkedList::DeleteMode, 37
  - constants::MenuLinkedList::SearchMode, 38
- TEXTBOX\_NAMES
  - constants::MenuArray::AddMode, 19
  - constants::MenuArray::CreateMode, 22
  - constants::MenuArray::UpdateMode, 26
  - constants::MenuDataStructure::CreateMode, 30
  - constants::MenuLinkedList::AddMode, 34
  - constants::MenuLinkedList::CreateMode, 36
  - constants::MenuLinkedList::UpdateMode, 39
- title
  - EventSquare, 119
- titleNodes
  - EventAnimation, 117
- titleWindow
  - constants, 13
- to\_string\_with\_precision
  - ToStringWithPrecision.hpp, 407
- toggle
  - Highlighter, 131
- toggleActiveColor
  - Arrow, 60
  - SingleNode, 214
- toggleActiveColorArrow
  - NodeInfo, 181
- toggleActiveColorNode
  - BackArrow, 65
  - NodeInfo, 181
- toggleLines
  - Array, 55
  - LinkedList, 139
- ToStringWithPrecision.hpp
  - to\_string\_with\_precision, 407
- transparentGreen
  - constants, 12
- Type
  - constants::MenuArray, 17
- type
  - pfd::internal::file\_dialog, 124
- TypeArray
  - Array, 48
- TypeLinkedList
  - LinkedList, 132
- TypeNode
  - NodeInfo, 175
- update
  - Array, 55
  - BaseScene, 69
  - Button, 73
  - CLLScene, 80
  - ControlMenu, 86
  - CustomTextbox, 88
  - DLLScene, 102
  - DynamicArrayScene, 111
  - LinkedList, 139
  - MainMenu, 143
  - MenuArray, 146
  - MenuDataStructure, 150
  - MenuLinkedList, 160
  - QueueScene, 196
  - SLLScene, 221
  - SquareInfo, 231
  - StackScene, 237
  - StaticArrayScene, 243
  - TextBox, 248
  - Window, 257
- UPDATE\_BUTTON
  - constants::MenuArray, 17
  - constants::MenuLinkedList, 32
- updateAddMode
  - MenuLinkedList, 161
- updateAnimation
  - Array, 55
  - LinkedList, 139
- updateArrows
  - NodeInfo, 181
- updateCreateMode
  - MenuLinkedList, 161
- updateDeleteMode
  - MenuLinkedList, 162
- updateModeEvents
  - CLLScene, 81
  - DLLScene, 104
  - DynamicArrayScene, 113
  - SLLScene, 223
  - StaticArrayScene, 245
- updateModeValue
  - MenuArray, 147
  - MenuLinkedList, 166
- updateMousePosition
  - MousePosition, 172
- updateNode
  - LinkedList, 140
  - NodeInfo, 182
- updateSearchMode
  - MenuLinkedList, 162
- updateSquare
  - Array, 56
- updateTextbox
  - MenuLinkedList, 166
- updateUpdateMode
  - MenuLinkedList, 163
- VALUE\_TEXTBOX
  - constants::MenuArray::AddMode, 19
  - constants::MenuArray::AllocateMode, 20
  - constants::MenuArray::SearchMode, 24

- constants::MenuArray::UpdateMode, [26](#)
- constants::MenuDataStructure::PushMode, [31](#)
- constants::MenuLinkedList::AddMode, [33](#)
- constants::MenuLinkedList::SearchMode, [38](#)
- constants::MenuLinkedList::UpdateMode, [39](#)
- Vector
  - Vector< T >, [249](#), [250](#)
- Vector< T >, [248](#)
  - ~Vector, [250](#)
  - assign, [250](#)
  - at, [250](#)
  - back, [251](#)
  - begin, [251](#)
  - clear, [251](#)
  - data, [251](#)
  - empty, [251](#)
  - end, [252](#)
  - erase, [252](#)
  - front, [252](#)
  - getCapacity, [253](#)
  - insert, [253](#)
  - operator=, [253](#)
  - operator[], [253](#)
  - pop\_back, [254](#)
  - push\_back, [254](#)
  - resize, [254](#)
  - size, [254](#)
  - Vector, [249](#), [250](#)
- vector\_result
  - pfd::internal::file\_dialog, [129](#)
- verbose
  - pfd::settings, [211](#)
- Visible
  - NodeInfo, [174](#)
- warning
  - pfd, [43](#)
- Window, [255](#)
  - ~Window, [256](#)
  - pollEvent, [256](#)
  - render, [257](#)
  - running, [257](#)
  - update, [257](#)
  - Window, [256](#)
- window
  - BaseDraw, [67](#)
  - BaseScene, [70](#)
  - MenuLinkedList, [166](#)
- yes
  - pfd, [42](#)
- yes\_no
  - pfd, [43](#)
- yes\_no\_cancel
  - pfd, [43](#)