

RocketRobot

Generated by Doxygen 1.8.6

Sun May 31 2015 17:57:21



# Contents

<b>1</b>	<b>Summary</b>	<b>1</b>
1.1	Overview . . . . .	1
1.1.1	Installation . . . . .	1
1.1.2	User Interface . . . . .	1
1.2	Neural network optimization . . . . .	2
1.3	Implementation . . . . .	2
1.3.1	Object motion . . . . .	2
1.3.2	Graphics . . . . .	2
1.3.3	Configuration system . . . . .	2
1.4	Licence . . . . .	3
<b>2</b>	<b>Namespace Index</b>	<b>5</b>
2.1	Namespace List . . . . .	5
<b>3</b>	<b>Hierarchical Index</b>	<b>7</b>
3.1	Class Hierarchy . . . . .	7
<b>4</b>	<b>Class Index</b>	<b>9</b>
4.1	Class List . . . . .	9
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11
<b>6</b>	<b>Namespace Documentation</b>	<b>13</b>
6.1	artist Namespace Reference . . . . .	13
6.1.1	Detailed Description . . . . .	13
6.1.2	Function Documentation . . . . .	13
6.1.2.1	debugArrow . . . . .	13
6.1.2.2	drawLight . . . . .	13
6.1.2.3	drawObject . . . . .	13
6.1.2.4	drawObstacle . . . . .	14
6.1.2.5	drawRobot . . . . .	14
6.1.2.6	drawSensor . . . . .	14

6.2	environment Namespace Reference	14
6.2.1	Detailed Description	15
6.2.2	Typedef Documentation	15
6.2.2.1	objectIterator	15
6.2.3	Function Documentation	15
6.2.3.1	addObject	15
6.2.3.2	clear	16
6.2.3.3	getCollisionId	16
6.2.3.4	getCollisionId	16
6.2.3.5	getCollisionId	16
6.2.3.6	getHitableCollisionId	17
6.2.3.7	getHitableCollisionId	17
6.2.3.8	getHitableCollisionId	17
6.2.3.9	getNumObjects	18
6.2.3.10	getObject	18
6.2.3.11	getObjectsBegin	18
6.2.3.12	getObjectsEnd	18
6.2.3.13	isColliding	18
6.2.3.14	isColliding	19
6.2.3.15	isCollidingWithHitable	19
6.2.3.16	isCollidingWithHitable	19
6.2.3.17	isTouchingHitableObject	20
6.2.3.18	isTouchingHitableObject	20
6.2.3.19	isTouchingHitableObject	20
6.2.3.20	isTouchingObject	20
6.2.3.21	isTouchingObject	21
6.2.3.22	isTouchingObject	21
6.2.3.23	isTouchingWall	21
6.2.3.24	isTouchingWall	21
6.2.3.25	removeObject	22
6.3	util Namespace Reference	22
6.3.1	Detailed Description	23
6.3.2	Function Documentation	23
6.3.2.1	addMovingLightSource	23
6.3.2.2	addNeuralNetworkRobotTarget	23
6.3.2.3	addObstacle	24
6.3.2.4	addRobot	24
6.3.2.5	addRobotTarget	24
6.3.2.6	addStationaryLightSource	25
6.3.2.7	advance	25

6.3.2.8	<a href="#">copy</a>	25
6.3.2.9	<a href="#">display</a>	26
6.3.2.10	<a href="#">getNumLights</a>	26
6.3.2.11	<a href="#">getNumObstacles</a>	26
6.3.2.12	<a href="#">getNumRobotsTargets</a>	26
6.3.2.13	<a href="#">newColor</a>	26
6.3.2.14	<a href="#">open</a>	26
6.3.2.15	<a href="#">removeAllLightSource</a>	26
6.3.2.16	<a href="#">removeAllObstacle</a>	26
6.3.2.17	<a href="#">removeAllRobotTarget</a>	26
6.3.2.18	<a href="#">removeLightSource</a>	27
6.3.2.19	<a href="#">removeObstacle</a>	27
6.3.2.20	<a href="#">removeRobotTarget</a>	27
6.3.2.21	<a href="#">reset</a>	27
6.3.2.22	<a href="#">save</a>	27
<b>7</b>	<b>Class Documentation</b>	<b>29</b>
7.1	<a href="#">BaseGfxApp Class Reference</a>	29
7.1.1	<a href="#">Detailed Description</a>	30
7.1.2	<a href="#">Constructor &amp; Destructor Documentation</a>	30
7.1.2.1	<a href="#">BaseGfxApp</a>	30
7.1.2.2	<a href="#">~BaseGfxApp</a>	30
7.1.3	<a href="#">Member Function Documentation</a>	30
7.1.3.1	<a href="#">advance</a>	30
7.1.3.2	<a href="#">display</a>	30
7.1.3.3	<a href="#">glui</a>	31
7.1.3.4	<a href="#">gluiControl</a>	31
7.1.3.5	<a href="#">graphicsBegin</a>	31
7.1.3.6	<a href="#">graphicsTimer</a>	31
7.1.3.7	<a href="#">handle</a>	31
7.1.3.8	<a href="#">height</a>	31
7.1.3.9	<a href="#">keyboard</a>	31
7.1.3.10	<a href="#">keyboardSpecial</a>	31
7.1.3.11	<a href="#">keyboardSpecialUp</a>	31
7.1.3.12	<a href="#">keyboardUp</a>	31
7.1.3.13	<a href="#">leftMouseDown</a>	31
7.1.3.14	<a href="#">leftMouseUp</a>	31
7.1.3.15	<a href="#">middleMouseDown</a>	31
7.1.3.16	<a href="#">middleMouseUp</a>	31
7.1.3.17	<a href="#">mouseDragged</a>	31

7.1.3.18	<a href="#">mouseMoved</a>	32
7.1.3.19	<a href="#">reshape</a>	32
7.1.3.20	<a href="#">rightMouseDown</a>	32
7.1.3.21	<a href="#">rightMouseUp</a>	32
7.1.3.22	<a href="#">runMainLoop</a>	32
7.1.3.23	<a href="#">s_draw</a>	32
7.1.3.24	<a href="#">s_glucallback</a>	32
7.1.3.25	<a href="#">s_keyboard</a>	32
7.1.3.26	<a href="#">s_keyboardspecial</a>	32
7.1.3.27	<a href="#">s_keyboardspecialup</a>	32
7.1.3.28	<a href="#">s_keyboardup</a>	32
7.1.3.29	<a href="#">s_mousebtn</a>	32
7.1.3.30	<a href="#">s_mousemotion</a>	32
7.1.3.31	<a href="#">s_reshape</a>	32
7.1.3.32	<a href="#">setCaption</a>	32
7.1.3.33	<a href="#">width</a>	32
7.1.4	<a href="#">Member Data Documentation</a>	32
7.1.4.1	<a href="#">m_drag</a>	32
7.1.4.2	<a href="#">m_glui</a>	32
7.1.4.3	<a href="#">m_glutWindowHandle</a>	32
7.1.4.4	<a href="#">m_height</a>	32
7.1.4.5	<a href="#">m_width</a>	32
7.1.4.6	<a href="#">s_currentApp</a>	32
7.1.4.7	<a href="#">s_glutInitialized</a>	33
7.2	<a href="#">Color Struct Reference</a>	33
7.2.1	<a href="#">Detailed Description</a>	33
7.2.2	<a href="#">Constructor &amp; Destructor Documentation</a>	33
7.2.2.1	<a href="#">Color</a>	33
7.2.2.2	<a href="#">Color</a>	33
7.2.2.3	<a href="#">Color</a>	34
7.2.2.4	<a href="#">Color</a>	34
7.2.3	<a href="#">Member Function Documentation</a>	34
7.2.3.1	<a href="#">isSimilar</a>	34
7.2.3.2	<a href="#">operator!=</a>	34
7.2.3.3	<a href="#">operator&lt;&lt;</a>	35
7.2.3.4	<a href="#">operator==</a>	35
7.2.4	<a href="#">Member Data Documentation</a>	35
7.2.4.1	<a href="#">blue</a>	35
7.2.4.2	<a href="#">green</a>	35
7.2.4.3	<a href="#">red</a>	35

7.3	ComplexRobot Class Reference	35
7.3.1	Detailed Description	36
7.3.2	Constructor & Destructor Documentation	36
7.3.2.1	ComplexRobot	36
7.3.2.2	ComplexRobot	36
7.3.2.3	ComplexRobot	36
7.3.2.4	~ComplexRobot	37
7.3.3	Member Function Documentation	37
7.3.3.1	getLeftSpeed	37
7.3.3.2	getRightSpeed	37
7.3.4	Member Data Documentation	37
7.3.4.1	defaultSpeed	37
7.3.4.2	enableLightSensors	37
7.3.4.3	enableObstacleSensors	37
7.3.4.4	enableRobotSensors	37
7.3.4.5	enableTargetSensors	37
7.3.4.6	lightSensorScale	37
7.3.4.7	lightSensorsCrossed	37
7.3.4.8	obstacleSensorScale	37
7.3.4.9	obstacleSensorsCrossed	37
7.3.4.10	robotSensorScale	37
7.3.4.11	robotSensorsCrossed	37
7.3.4.12	targetSensorScale	37
7.3.4.13	targetSensorsCrossed	37
7.4	LightSource Class Reference	37
7.4.1	Detailed Description	38
7.4.2	Constructor & Destructor Documentation	38
7.4.2.1	LightSource	38
7.4.2.2	LightSource	38
7.4.2.3	LightSource	39
7.4.2.4	~LightSource	39
7.4.3	Member Function Documentation	39
7.4.3.1	display	39
7.4.3.2	handleCollision	39
7.5	Location Struct Reference	40
7.5.1	Detailed Description	40
7.5.2	Constructor & Destructor Documentation	40
7.5.2.1	Location	40
7.5.2.2	Location	40
7.5.3	Member Function Documentation	40

7.5.3.1	operator==	40
7.5.4	Member Data Documentation	40
7.5.4.1	x	40
7.5.4.2	y	40
7.6	NeuralNetwork Class Reference	41
7.6.1	Detailed Description	41
7.6.2	Constructor & Destructor Documentation	41
7.6.2.1	NeuralNetwork	41
7.6.2.2	NeuralNetwork	41
7.6.3	Member Function Documentation	41
7.6.3.1	combine	41
7.6.3.2	compute	42
7.6.3.3	load	43
7.6.3.4	mutate	43
7.6.3.5	write	43
7.7	NeuralNetworkRobot Class Reference	43
7.7.1	Detailed Description	44
7.7.2	Constructor & Destructor Documentation	44
7.7.2.1	NeuralNetworkRobot	44
7.7.2.2	NeuralNetworkRobot	44
7.7.2.3	~NeuralNetworkRobot	44
7.7.3	Member Function Documentation	44
7.7.3.1	getLeftSpeed	44
7.7.3.2	getRightSpeed	45
7.7.4	Member Data Documentation	45
7.7.4.1	filename	45
7.8	NoOpenLocationException Class Reference	45
7.8.1	Detailed Description	45
7.8.2	Constructor & Destructor Documentation	45
7.8.2.1	NoOpenLocationException	45
7.9	ObjectIterator Class Reference	45
7.9.1	Detailed Description	46
7.9.2	Constructor & Destructor Documentation	46
7.9.2.1	ObjectIterator	46
7.9.2.2	ObjectIterator	46
7.9.2.3	~ObjectIterator	46
7.9.3	Member Function Documentation	46
7.9.3.1	operator!=	46
7.9.3.2	operator*	46
7.9.3.3	operator++	46



7.9.3.4	<a href="#">operator++</a>	47
7.9.3.5	<a href="#">operator&lt;</a>	47
7.9.3.6	<a href="#">operator==</a>	47
7.9.3.7	<a href="#">operator&gt;</a>	47
7.9.4	<a href="#">Friends And Related Function Documentation</a>	47
7.9.4.1	<a href="#">environment::getObjectsBegin</a>	47
7.9.4.2	<a href="#">environment::getObjectsEnd</a>	47
7.10	<a href="#">Obstacle Class Reference</a>	47
7.10.1	<a href="#">Detailed Description</a>	47
7.10.2	<a href="#">Constructor &amp; Destructor Documentation</a>	47
7.10.2.1	<a href="#">Obstacle</a>	47
7.10.2.2	<a href="#">Obstacle</a>	48
7.10.2.3	<a href="#">Obstacle</a>	48
7.10.2.4	<a href="#">~Obstacle</a>	48
7.10.3	<a href="#">Member Function Documentation</a>	48
7.10.3.1	<a href="#">handleCollision</a>	48
7.11	<a href="#">OptimizeSimulation Class Reference</a>	49
7.11.1	<a href="#">Detailed Description</a>	49
7.11.2	<a href="#">Constructor &amp; Destructor Documentation</a>	49
7.11.2.1	<a href="#">OptimizeSimulation</a>	49
7.11.2.2	<a href="#">~OptimizeSimulation</a>	49
7.11.3	<a href="#">Member Function Documentation</a>	49
7.11.3.1	<a href="#">getPerformance</a>	50
7.11.3.2	<a href="#">getPerformanceMaze</a>	50
7.11.3.3	<a href="#">getPerformanceObstacles</a>	50
7.11.3.4	<a href="#">getPerformanceRandomRepeated</a>	50
7.11.3.5	<a href="#">runMainLoop</a>	50
7.11.3.6	<a href="#">stop</a>	50
7.12	<a href="#">PhysicalObject Class Reference</a>	50
7.12.1	<a href="#">Detailed Description</a>	51
7.12.2	<a href="#">Constructor &amp; Destructor Documentation</a>	51
7.12.2.1	<a href="#">PhysicalObject</a>	51
7.12.2.2	<a href="#">PhysicalObject</a>	51
7.12.2.3	<a href="#">PhysicalObject</a>	51
7.12.2.4	<a href="#">~PhysicalObject</a>	51
7.12.3	<a href="#">Member Function Documentation</a>	51
7.12.3.1	<a href="#">display</a>	51
7.12.3.2	<a href="#">findOpenLocation</a>	52
7.12.3.3	<a href="#">forceSetPosition</a>	53
7.12.3.4	<a href="#">forceTranslate</a>	53

7.12.3.5	<a href="#">getColor</a>	53
7.12.3.6	<a href="#">getId</a>	53
7.12.3.7	<a href="#">getLocation</a>	53
7.12.3.8	<a href="#">getOrientation</a>	53
7.12.3.9	<a href="#">getRadius</a>	54
7.12.3.10	<a href="#">getSpeed</a>	54
7.12.3.11	<a href="#">getXPosition</a>	54
7.12.3.12	<a href="#">getYPosition</a>	54
7.12.3.13	<a href="#">handleCollision</a>	54
7.12.3.14	<a href="#">hasEqualPosition</a>	54
7.12.3.15	<a href="#">pointTo</a>	55
7.12.3.16	<a href="#">reorient</a>	55
7.12.3.17	<a href="#">rotate</a>	55
7.12.3.18	<a href="#">setColor</a>	55
7.12.3.19	<a href="#">setLocation</a>	55
7.12.3.20	<a href="#">setOrientation</a>	56
7.12.3.21	<a href="#">setPosition</a>	56
7.12.3.22	<a href="#">setRadius</a>	56
7.12.3.23	<a href="#">setSpeed</a>	56
7.12.3.24	<a href="#">translate</a>	56
7.12.3.25	<a href="#">update</a>	57
7.12.3.26	<a href="#">updateMembers</a>	57
7.12.3.27	<a href="#">updatePosition</a>	57
7.12.4	<a href="#">Member Data Documentation</a>	57
7.12.4.1	<a href="#">isHitable</a>	57
7.12.4.2	<a href="#">objectType</a>	57
7.13	<a href="#">Robot Class Reference</a>	57
7.13.1	<a href="#">Detailed Description</a>	58
7.13.2	<a href="#">Constructor &amp; Destructor Documentation</a>	58
7.13.2.1	<a href="#">Robot</a>	58
7.13.2.2	<a href="#">Robot</a>	58
7.13.2.3	<a href="#">~Robot</a>	59
7.13.3	<a href="#">Member Function Documentation</a>	59
7.13.3.1	<a href="#">display</a>	59
7.13.3.2	<a href="#">getLeftSpeed</a>	59
7.13.3.3	<a href="#">getLineColor</a>	59
7.13.3.4	<a href="#">getRightSpeed</a>	59
7.13.3.5	<a href="#">getTarget</a>	59
7.13.3.6	<a href="#">handleCollision</a>	60
7.13.3.7	<a href="#">setLineColor</a>	60

7.13.3.8	setTarget	60
7.13.3.9	update	60
7.13.3.10	updateMembers	60
7.13.4	Member Data Documentation	61
7.13.4.1	robotType	61
7.14	Sensor Class Reference	61
7.14.1	Constructor & Destructor Documentation	61
7.14.1.1	Sensor	61
7.14.1.2	Sensor	61
7.14.2	Member Function Documentation	62
7.14.2.1	display	62
7.14.2.2	getOrientation	62
7.14.2.3	getPosition	62
7.14.2.4	getTypeDetected	62
7.14.2.5	getViewAngle	62
7.14.2.6	getXPosition	63
7.14.2.7	getYPosition	63
7.14.2.8	sense	63
7.14.2.9	setOrientation	63
7.14.2.10	setPosition	63
7.14.2.11	setPosition	63
7.14.2.12	setTypeDetected	64
7.14.2.13	setViewAngle	64
7.14.2.14	updatePosition	64
7.15	SimpleRobot Class Reference	64
7.15.1	Detailed Description	65
7.15.2	Constructor & Destructor Documentation	65
7.15.2.1	SimpleRobot	65
7.15.2.2	SimpleRobot	65
7.15.2.3	~SimpleRobot	65
7.15.3	Member Function Documentation	65
7.15.3.1	getLeftSpeed	65
7.15.3.2	getRightSpeed	65
7.16	Simulation Class Reference	65
7.16.1	Detailed Description	67
7.16.2	Constructor & Destructor Documentation	67
7.16.2.1	Simulation	67
7.16.2.2	~Simulation	68
7.16.3	Member Function Documentation	68
7.16.3.1	advance	68

7.16.3.2	clear	68
7.16.3.3	display	68
7.16.3.4	gluiControl	68
7.16.3.5	initObjects	68
7.16.3.6	keyboard	68
7.16.3.7	keyboardSpecial	68
7.16.3.8	leftMouseDown	68
7.16.3.9	leftMouseUp	68
7.16.3.10	middleMouseDown	69
7.16.3.11	mouseDragged	69
7.16.3.12	pause	69
7.16.3.13	random	69
7.16.3.14	reset	69
7.16.3.15	resume	69
7.16.3.16	s_addMovingLightSource	69
7.16.3.17	s_addObstacle	69
7.16.3.18	s_addRobot	69
7.16.3.19	s_addRobotTarget	69
7.16.3.20	s_addStationaryLightSource	69
7.16.3.21	s_clear	69
7.16.3.22	s_neuralNetworkFileChanged	69
7.16.3.23	s_open	69
7.16.3.24	s_pause	69
7.16.3.25	s_random	69
7.16.3.26	s_refreshConfiguration	69
7.16.3.27	s_removeAllLightSource	69
7.16.3.28	s_removeAllObstacle	69
7.16.3.29	s_removeAllRobotTarget	69
7.16.3.30	s_removeLightSource	69
7.16.3.31	s_removeObstacle	69
7.16.3.32	s_removeRobotTarget	69
7.16.3.33	s_reset	69
7.16.3.34	s_resume	70
7.16.3.35	s_save	70
7.16.3.36	s_start	70
7.16.3.37	start	70
7.16.3.38	tryAddMovingLightSource	70
7.16.3.39	tryAddObstacle	70
7.16.3.40	tryAddRobot	70
7.16.3.41	tryAddRobotTarget	70

7.16.3.42 tryAddStationaryLightSource . . . . .	70
7.16.3.43 tryOpen . . . . .	70
7.16.3.44 tryRemoveLightSource . . . . .	70
7.16.3.45 tryRemoveObstacle . . . . .	70
7.16.3.46 tryRemoveRobotTarget . . . . .	70
7.16.3.47 trySave . . . . .	70
7.17 Target Class Reference . . . . .	71
7.17.1 Detailed Description . . . . .	71
7.17.2 Constructor & Destructor Documentation . . . . .	71
7.17.2.1 Target . . . . .	71
7.17.2.2 Target . . . . .	71
7.17.2.3 Target . . . . .	71
7.17.2.4 Target . . . . .	71
7.17.2.5 ~Target . . . . .	71
7.17.3 Member Function Documentation . . . . .	71
7.17.3.1 handleCollision . . . . .	71
<b>8 File Documentation</b>	<b>73</b>
8.1 artist.cpp File Reference . . . . .	73
8.1.1 Macro Definition Documentation . . . . .	73
8.1.1.1 _USE_MATH_DEFINES . . . . .	73
8.2 artist.h File Reference . . . . .	73
8.2.1 Detailed Description . . . . .	74
8.3 BaseGfxApp.cpp File Reference . . . . .	74
8.3.1 Detailed Description . . . . .	74
8.4 BaseGfxApp.h File Reference . . . . .	74
8.4.1 Detailed Description . . . . .	75
8.5 Color.cpp File Reference . . . . .	75
8.5.1 Detailed Description . . . . .	75
8.6 Color.h File Reference . . . . .	75
8.6.1 Detailed Description . . . . .	76
8.6.2 Macro Definition Documentation . . . . .	76
8.6.2.1 GET_COLOR . . . . .	76
8.6.3 Typedef Documentation . . . . .	76
8.6.3.1 Color . . . . .	76
8.7 ComplexRobot.cpp File Reference . . . . .	76
8.7.1 Detailed Description . . . . .	76
8.8 ComplexRobot.h File Reference . . . . .	76
8.8.1 Detailed Description . . . . .	77
8.9 environment.cpp File Reference . . . . .	77

8.9.1	Function Documentation	77
8.9.1.1	isTouching	77
8.10	environment.h File Reference	77
8.11	LightSource.cpp File Reference	79
8.11.1	Detailed Description	79
8.12	LightSource.h File Reference	79
8.12.1	Detailed Description	79
8.13	Location.h File Reference	79
8.13.1	Typedef Documentation	80
8.13.1.1	Location	80
8.14	main.cpp File Reference	80
8.14.1	Detailed Description	80
8.14.2	Macro Definition Documentation	80
8.14.2.1	DEFAULT_CONFIG	80
8.14.3	Function Documentation	80
8.14.3.1	main	80
8.15	mainpage.h File Reference	80
8.16	NeuralNetwork.cpp File Reference	80
8.16.1	Detailed Description	81
8.17	NeuralNetwork.h File Reference	81
8.17.1	Detailed Description	81
8.18	NeuralNetworkRobot.cpp File Reference	81
8.18.1	Detailed Description	82
8.19	NeuralNetworkRobot.h File Reference	82
8.19.1	Detailed Description	82
8.20	Obstacle.cpp File Reference	82
8.20.1	Detailed Description	82
8.21	Obstacle.h File Reference	83
8.21.1	Detailed Description	83
8.22	OptimizeSimulation.cpp File Reference	83
8.23	OptimizeSimulation.h File Reference	84
8.23.1	Detailed Description	84
8.24	PhysicalObject.cpp File Reference	84
8.24.1	Detailed Description	84
8.24.2	Macro Definition Documentation	85
8.24.2.1	_USE_MATH_DEFINES	85
8.25	PhysicalObject.h File Reference	85
8.25.1	Detailed Description	85
8.25.2	Enumeration Type Documentation	85
8.25.2.1	ObjectType	85

8.26 Robot.cpp File Reference . . . . .	85
8.26.1 Detailed Description . . . . .	86
8.27 Robot.h File Reference . . . . .	86
8.27.1 Detailed Description . . . . .	86
8.27.2 Enumeration Type Documentation . . . . .	86
8.27.2.1 RobotType . . . . .	86
8.28 Sensor.cpp File Reference . . . . .	86
8.28.1 Detailed Description . . . . .	87
8.28.2 Macro Definition Documentation . . . . .	87
8.28.2.1 _USE_MATH_DEFINES . . . . .	87
8.29 Sensor.h File Reference . . . . .	87
8.29.1 Detailed Description . . . . .	87
8.30 SimpleRobot.cpp File Reference . . . . .	87
8.30.1 Detailed Description . . . . .	88
8.31 SimpleRobot.h File Reference . . . . .	88
8.31.1 Detailed Description . . . . .	88
8.32 Simulation.cpp File Reference . . . . .	89
8.32.1 Detailed Description . . . . .	89
8.32.2 Function Documentation . . . . .	89
8.32.2.1 ftoS . . . . .	89
8.33 Simulation.h File Reference . . . . .	89
8.33.1 Detailed Description . . . . .	90
8.34 Target.cpp File Reference . . . . .	90
8.34.1 Detailed Description . . . . .	90
8.35 Target.h File Reference . . . . .	90
8.35.1 Detailed Description . . . . .	91
8.36 util.cpp File Reference . . . . .	91
8.36.1 Detailed Description . . . . .	91
8.36.2 Variable Documentation . . . . .	92
8.36.2.1 colorNum . . . . .	92
8.36.2.2 lights . . . . .	92
8.36.2.3 objectsMutex . . . . .	92
8.36.2.4 obstacles . . . . .	92
8.36.2.5 robots . . . . .	92
8.36.2.6 targets . . . . .	92
8.37 util.h File Reference . . . . .	92
8.37.1 Detailed Description . . . . .	93





# Chapter 1

## Summary

### 1.1 Overview

RocketRobot is a multi-featured robot simulator. It simulates robots, targets, and obstacles in an environment, with the goal of robots seeking and finding their targets. When a robot finds its target, both disappear from the simulation. In addition, stimuli or light sources can be placed in the environment. There are several types of robots. Simple robots have sensors that only seek the target, and ignore obstacles. Complex robots have sensors for lights, robots, obstacles, and the target. Note that obstacle sensors include the wall as an 'obstacle', straight ahead of the sensor. For both robot types, the wheels speeds are controlled directly by the sensor readings. Complex robots can have each type of sensor scaled or connected in any pattern, this is controllable from the 'New complex robot settings' tab in the interface. Neural network robots use a feed-forward neural network for control. The network takes the target, robot, and obstacle settings as inputs, and outputs the wheel speeds. The network is loaded from a file, specified in the 'New neural network robot settings' tab in the interface.

#### 1.1.1 Installation

You can execute the 'gorobot' executable directly from the src or bin folder, but it is easier to directly install it. This allows you to run it as 'rocketrobot <optional filename>=>' from any folder. To install on a Linux computer, simply run the install script from the main folder. This soft-links a script to your ~/bin folder. This requires that you have a ~/bin folder on your path, if not you can create one and add it to your .bashrc or .cshrc. To uninstall, you can simply run the uninstall script.

#### 1.1.2 User Interface

The RocketRobot user interface is fairly simple. The control panel allows you to start, stop, pause, resume, and quit the simulation. The reset button reverts to the previous time it was started if the simulation was opened from a file, or randomly generates a new simulation if the current simulation is random. The 'Refresh settings' button reloads the configuration files (described later). The open/save tab also lets you open or save the current state. You can also provide a file to open as the command-line argument. There are several examples simulations for this in the examples folder.

The user interface is fairly straightforward for adding and removing objects. There are several control tabs from which the number of obstacles, robots, and lights can be controlled. Objects are added and removed in a FILO manner from the interface. Targets are always created with a robot, but not all robots need to have a target. Decreasing the number of robots through the control panel deletes the target paired with the removed robot.

For more information about an object (location, speed, orientation, radius), you can click on an object. You can also drag and drop objects to move them, or drag them off the screen to delete them. The radius of the selected object can be changed with the up and down arrow keys, the orientation with the left and right keys, and the speed with the + and - keys. You can also middle-click to paste a copy of the selected object. Note that copying a robot creates a new robot with the same target (if any.) You cannot copy targets.

## 1.2 Neural network optimization

A major portion of the project is the neural network-driven robot, and the associated optimization algorithms. To optimize the neural network robot, a separate simulation class, [OptimizeSimulation](#), was created. Optimization employs a genetic algorithm, in which a pool of possible networks is maintained, along with their performance. The performances are calculated by performing a large number of runs of (consistent) random simulations, and summing the total time for all robots to find all targets. In addition, several specific test cases are run for each network to test some interesting cases (e.g. navigating a simple maze, etc.) Pairs of networks are randomly selected from the pool (with one network from a better subset of the pool.) These are combined by taking half the weights from each and merging them into a new network, then making more random tweaks. This is then tested and added to the pool if it is better than the worst performance in the pool. The best performance in the pool is the optimal network.

To run the optimization, run `./optimize` from the scripts folder. The temporary and intermediate files get saved to `runtime/neuralnetwork`. This includes the optimization log, pool performances, optimal network, pool networks, and several others.

## 1.3 Implementation

The various objects are all subclasses of [PhysicalObject](#), which handles basic behaviors and attributes. This class is virtual, so that specific behaviors for collisions, etc are handled individually, and can optionally call back to the default handlers in [PhysicalObject](#). Each type of robot is a subclass of [Robot](#), which is also virtual, as it has virtual functions to get new wheel speeds given the sensor readings. Everything else, including reading from sensors and collision behavior, is handled directly by the robot class.

There is also an environment namespace with which all objects register, and are automatically removed when they are destructed. An iterator over all objects can be requested from the environment, and they can also be accessed by id. The environment also contains functions for detecting collisions between walls and objects. A util namespace provides an easy way of adding and removing objects, managing stacks so that objects are added and removed in FILO order. It also contains functionality to open and save the current state to and from files.

The [Simulation](#) class handles the user interface, providing a wrapper for the functions in util. It also acts as a driver, requesting updates and redraws from all objects.

### 1.3.1 Object motion

Object movement is handled by [PhysicalObject](#). It has an `updateMembers` function which advances all objects by a distance calculated from the speed, after calling the virtual update function. The robot class update sets the position and speed from the calculated wheel speeds. The `updateMembers` function is called for each object from `advance` in util, which in turn is called from `advance` in [Simulation](#) which is triggered by a glut timer function.

### 1.3.2 Graphics

A graphical display of the simulation is rendered by OpenGL. Most of the graphics functionality is in the artist namespace, which has simple functions for drawing circles and lines, which are used by more complex functions for different kinds of objects. There is a virtual function in [PhysicalObject](#) to render each object as a circle, and this is overridden by [Robot](#) and [LightSource](#). These are called by simulation similarly to `updatePosition`.

### 1.3.3 Configuration system

Most settings are loaded through the configuration system, see `configuration.h` for more details. This allows settings to be stored externally, and avoids needing to recompile often when making small changes. Configuration values can also be set from the command line.

## 1.4 Licence

This project has been developed by Lucas Kramer, Carl Bahn, Himawan Go, and Xi Zhang. It has been extended by Lucas Kramer to add neural network-controlled robots and made the interface more useable. Copyright (C) 2015 Lucas Kramer, Carl Bahn, Himawan Go, and Xi Zhang

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">artist</a>	Graphical commands . . . . .	<a href="#">13</a>
<a href="#">environment</a>	Environment namespace, handles all the objects as a group. Also manages object ids and collision detection . . . . .	<a href="#">14</a>
<a href="#">util</a>	Util namespace, contains helper functions to add and remove robots . . . . .	<a href="#">22</a>



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

BaseGfxApp . . . . .	29
Simulation . . . . .	65
Color . . . . .	33
Location . . . . .	40
NeuralNetwork . . . . .	41
ObjectIterator . . . . .	45
OptimizeSimulation . . . . .	49
PhysicalObject . . . . .	50
LightSource . . . . .	37
Obstacle . . . . .	47
Robot . . . . .	57
ComplexRobot . . . . .	35
NeuralNetworkRobot . . . . .	43
SimpleRobot . . . . .	64
Target . . . . .	71
runtime_error	
NoOpenLocationException . . . . .	45
Sensor . . . . .	61





## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">BaseGfxApp</a>	Graphics driver class . . . . .	29
<a href="#">Color</a>	This struct holds the representation of a color . . . . .	33
<a href="#">ComplexRobot</a>	A complex robot with configurable feedback from all sensors . . . . .	35
<a href="#">LightSource</a>	<a href="#">LightSource</a> for the robot to seek . . . . .	37
<a href="#">Location</a>	Xy position on the screen, approximately in pixels x is horizontal from the left and y is vertical from the bottom . . . . .	40
<a href="#">NeuralNetwork</a>	A representation of a neural network . . . . .	41
<a href="#">NeuralNetworkRobot</a>	A robot controlled by a neural network with inputs from the sensors . . . . .	43
<a href="#">NoOpenLocationException</a>	An exception to be thrown when an open <a href="#">Location</a> cannot be found . . . . .	45
<a href="#">ObjectIterator</a>	<a href="#">ObjectIterator</a> class, a simple iterator for the object in environment . . . . .	45
<a href="#">Obstacle</a>	<a href="#">Obstacle</a> for the robot to hit/avoid . . . . .	47
<a href="#">OptimizeSimulation</a>	<a href="#">OptimizeSimulation</a> class, sets up environments and robots . . . . .	49
<a href="#">PhysicalObject</a>	This is a common superclass for all objects . . . . .	50
<a href="#">Robot</a>	<a href="#">Robot</a> that moves around the window and bumps into obstacles . . . . .	57
<a href="#">Sensor</a>	. . . . .	61
<a href="#">SimpleRobot</a>	A simple robot with uncrossed feedback . . . . .	64
<a href="#">Simulation</a>	<a href="#">Simulation</a> class, sets up the GUI and the drawing environment . . . . .	65
<a href="#">Target</a>	<a href="#">Target</a> for the robot to seek . . . . .	71



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">artist.cpp</a>	73
<a href="#">artist.h</a>	
A namespace for OpenGL graphical functions	73
<a href="#">BaseGfxApp.cpp</a>	
Implementation of <a href="#">BaseGfxApp</a>	74
<a href="#">BaseGfxApp.h</a>	
The basic application class for CSci-3081 project. Uses GLUT and GLUI and wraps them in a nice C++ interface	74
<a href="#">Color.cpp</a>	
Utility functions for handling color	75
<a href="#">Color.h</a>	
Definintions for representing any hexadecimal color	75
<a href="#">ComplexRobot.cpp</a>	
The representation of a robot within the simulation	76
<a href="#">ComplexRobot.h</a>	
The representation of a robot within the simulation	76
<a href="#">environment.cpp</a>	77
<a href="#">environment.h</a>	77
<a href="#">LightSource.cpp</a>	
The representation of a target in the simulation	79
<a href="#">LightSource.h</a>	
The representation of an obstacle in the simulation	79
<a href="#">Location.h</a>	79
<a href="#">main.cpp</a>	
The main function to execute the simulation program	80
<a href="#">mainpage.h</a>	80
<a href="#">NeuralNetwork.cpp</a>	
A feed-forward neural network class	80
<a href="#">NeuralNetwork.h</a>	
A feed-forward neural network class	81
<a href="#">NeuralNetworkRobot.cpp</a>	
The representation of a neural network robot within the simulation	81
<a href="#">NeuralNetworkRobot.h</a>	
The representation of a neural network robot	82
<a href="#">Obstacle.cpp</a>	
The representation of an obstacle in the simulation	82
<a href="#">Obstacle.h</a>	
The representation of an obstacle in the simulation	83

<a href="#">OptimizeSimulation.cpp</a> . . . . .	83
<a href="#">OptimizeSimulation.h</a>	
<a href="#">Robot</a> simultaion for optimizing the <a href="#">NeuralNetworkRobot</a> . . . . .	84
<a href="#">PhysicalObject.cpp</a>	
The representation of any object within the simulation . . . . .	84
<a href="#">PhysicalObject.h</a>	
The representation of any object within the simulation . . . . .	85
<a href="#">Robot.cpp</a>	
The representation of robot within the simulation . . . . .	85
<a href="#">Robot.h</a>	
The representation of robot within the simulation . . . . .	86
<a href="#">Sensor.cpp</a>	
A sensor that detects certain objects in the environment . . . . .	86
<a href="#">Sensor.h</a>	
A sensor that detects certain objects in the environment . . . . .	87
<a href="#">SimpleRobot.cpp</a>	
The representation of robot within the simulation . . . . .	87
<a href="#">SimpleRobot.h</a>	
The representation of a simple robot . . . . .	88
<a href="#">Simulation.cpp</a>	
Implementation for the main application class of the robot simulation . . . . .	89
<a href="#">Simulation.h</a>	
Main application class for the robot simulation . . . . .	89
<a href="#">Target.cpp</a>	
The representation of a target in the simulation . . . . .	90
<a href="#">Target.h</a>	
The representation of an obstacle in the simulation . . . . .	90
<a href="#">util.cpp</a>	
Implmentation of functions for the simulation . . . . .	91
<a href="#">util.h</a>	
Helper functions for the simulation . . . . .	92

## Chapter 6

# Namespace Documentation

### 6.1 artist Namespace Reference

graphical commands

#### Functions

- void [drawObject](#) ([Location](#) loc, int radius, [Color](#) color)
- void [debugArrow](#) ([Location](#) loc, int orientation)
- void [drawLight](#) ([Location](#) loc, int radius, [Color](#) color)
- void [drawObstacle](#) ([Location](#) loc, int radius)
- void [drawSensor](#) ([Location](#) loc, int orientation, int angle, float intensity)
- void [drawRobot](#) ([Location](#) loc, int radius, int orientation, [Color](#) color, [Color](#) lineColor)

#### 6.1.1 Detailed Description

graphical commands

##### Author

Carl artist functions provide a contained way to display graphics without putting OpenGL graphics in each file

#### 6.1.2 Function Documentation

6.1.2.1 void artist::debugArrow ( [Location](#) *loc*, int *orientation* )

6.1.2.2 void artist::drawLight ( [Location](#) *loc*, int *radius*, [Color](#) *color* )

Draws a [LightSource](#)

##### Parameters

<i>loc</i>	absolute <a href="#">Location</a> to draw center of light
<i>radius</i>	radius of 'bulb' of light

6.1.2.3 void artist::drawObject ( [Location](#) *loc*, int *radius*, [Color](#) *color* )

**Author**

Lucas Kramer Draws the default Object, which is a circle

**Parameters**

<i>loc</i>	absolute <a href="#">Location</a> to draw center of circle
<i>radius</i>	radius of circle
<i>color</i>	the color of the circle

**6.1.2.4 void artist::drawObstacle ( [Location](#) *loc*, int *radius* )**

Draws an obstacle

**Parameters**

<i>loc</i>	absolute <a href="#">Location</a> to draw center of <a href="#">Obstacle</a>
<i>radius</i>	radius of <a href="#">Obstacle</a>

**6.1.2.5 void artist::drawRobot ( [Location](#) *loc*, int *radius*, int *orientation*, [Color](#) *color*, [Color](#) *lineColor* )**

Draws a [Robot](#)

**Parameters**

<i>loc</i>	absolute <a href="#">Location</a> of center of robot
<i>orientation</i>	absolute direction for robot to face
<i>color</i>	the color of the robot's body
<i>lineColor</i>	the color of the robot's direction line

**6.1.2.6 void artist::drawSensor ( [Location](#) *loc*, int *orientation*, int *angle*, float *intensity* )**

Draws a sensor

**Parameters**

<i>loc</i>	absolute <a href="#">Location</a> to draw center of <a href="#">Sensor</a>
<i>orientation</i>	absolute direction of sensor
<i>angle</i>	number of degrees that sensor senses in, with orientation in center
<i>intensity</i>	redness of sensor as value between 0 and 1, meant to indicate amount of light detected

## 6.2 environment Namespace Reference

environment namespace, handles all the objects as a group. Also manages object ids and collision detection

**Typedefs**

- typedef [ObjectIterator](#) [objectIterator](#)

**Functions**

- int [addObject](#) ([PhysicalObject](#) \*object)  
Adds an object to the environment.

- void `removeObject` (int id)  
*Removes an object from the environment.*
- void `clear` ()  
*Removes all objects from the environment and resets the id counter.*
- unsigned `getNumObjects` ()  
*Gets the number of objects in environment.*
- `PhysicalObject` \* `getObject` (int id)  
*Finds an object from the environment.*
- `objectIterator` `getObjectsBegin` ()  
*Gets an iterator to the beginning of the objects.*
- `objectIterator` `getObjectsEnd` ()  
*Gets an iterator to the end of the objects.*
- bool `isTouchingWall` (`Location` l, int r)
- bool `isTouchingWall` (int id)
- bool `isTouchingObject` (`Location` l, int r, int id)
- bool `isTouchingHitableObject` (`Location` l, int r, int id)
- bool `isTouchingObject` (`Location` l, int r)
- bool `isTouchingHitableObject` (`Location` l, int r)
- bool `isTouchingObject` (int id)
- bool `isTouchingHitableObject` (int id)
- bool `isColliding` (`Location` l, int r)
- bool `isColliding` (int id)
- bool `isCollidingWithHitable` (`Location` l, int r)
- bool `isCollidingWithHitable` (int id)
- int `getCollisionId` (`Location` l, int r, int id)
- int `getHitableCollisionId` (`Location` l, int r, int id)
- int `getCollisionId` (`Location` l, int r)
- int `getHitableCollisionId` (`Location` l, int r)
- int `getCollisionId` (int id)
- int `getHitableCollisionId` (int id)

### 6.2.1 Detailed Description

environment namespace, handles all the objects as a group. Also manages object ids and collision detection

### 6.2.2 Typedef Documentation

#### 6.2.2.1 typedef `ObjectIterator` `environment::objectIterator`

An object iterator

### 6.2.3 Function Documentation

#### 6.2.3.1 int `environment::addObject` ( `PhysicalObject` \* *object* )

Adds an object to the environment.

Parameters

---

<i>object</i>	The object to add
---------------	-------------------

**Returns**

The assigned id of the object

**6.2.3.2 void environment::clear ( )**

Removes all objects from the environment and resets the id counter.

**6.2.3.3 int environment::getCollisionId ( Location *l*, int *r*, int *id* )**

Determines what object is being hit by another object

**Parameters**

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object
<i>id</i>	The id of the object to exclude

**Returns**

The id of one object that is being hit, or -1 if it is hitting nothing or a wall

**See Also**

[isColliding](#)

**6.2.3.4 int environment::getCollisionId ( Location *l*, int *r* )**

Determines what object is being hit by another object

**Parameters**

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object

**Returns**

The id of one object that is being hit, or -1 if it is hitting nothing or a wall

**See Also**

[isColliding](#)

**6.2.3.5 int environment::getCollisionId ( int *id* )**

Determines what object is being hit by another object



## Parameters

<i>id</i>	The id of the object
-----------	----------------------

## Returns

The id of one object that is being hit, or -1 if it is hiting nothing or a wall

## See Also

[isColliding](#)

**6.2.3.6** `int environment::getHitableCollisionId ( Location l, int r, int id )`

Determines what hitable object is being hit by another object

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object
<i>id</i>	The id of the object to exclude

## Returns

The id of one hitable object that is being hit, or -1 if it is hiting nothing or a wall

## See Also

[isColliding](#)

**6.2.3.7** `int environment::getHitableCollisionId ( Location l, int r )`

Determines what hitable object is being hit by another object

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object

## Returns

The id of one hitable object that is being hit, or -1 if it is hiting nothing or a wall

## See Also

[isColliding](#)

**6.2.3.8** `int environment::getHitableCollisionId ( int id )`

Determines what hitable object is being hit by another object

## Parameters

<i>id</i>	The id of the object
-----------	----------------------

## Returns

The id of one hitable object that is being hit, or -1 if it is hiting nothing or a wall

## See Also

[isColliding](#)

**6.2.3.9 unsigned environment::getNumObjects ( )**

Gets the number of objects in environment.

## Returns

The number of objects

**6.2.3.10 PhysicalObject \* environment::getObject ( int *id* )**

Finds an object from the environment.

## Parameters

<i>id</i>	The id of the object to find
-----------	------------------------------

## Returns

The object that was looked iup

**6.2.3.11 objectIterator environment::getObjectsBegin ( )**

Gets an iterator to the beginning of the objects.

## Returns

The iterator

**6.2.3.12 objectIterator environment::getObjectsEnd ( )**

Gets an iterator to the end of the objects.

## Returns

The iterator

**6.2.3.13 bool environment::isColliding ( Location *l*, int *r* )**

Checks if the object specified by the given [Location](#) and radius is touching a wall or any other objects

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object

## Returns

true when the object is touching a wall or any other object

## See Also

[isTouchingWall](#), [isTouchingObject](#)

**6.2.3.14** `bool environment::isColliding ( int id )`

Checks if the object specified by the given *id* is touching a wall or any other objects

## Parameters

<i>id</i>	The id of the object
-----------	----------------------

## Returns

true when the object is touching a wall or any other object

## See Also

[isTouchingWall](#), [isTouchingObject](#)

**6.2.3.15** `bool environment::isCollidingWithHitable ( Location l, int r )`

Checks if the object specified by the given [Location](#) and radius is touching a wall or any other hitable objects

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object

## Returns

true when the object is touching a wall or any other hitable object

## See Also

[isTouchingWall](#), [isTouchingObject](#)

**6.2.3.16** `bool environment::isCollidingWithHitable ( int id )`

Checks if the object specified by the given *id* is touching a wall or any other hitable objects

## Parameters

<i>id</i>	The id of the object
-----------	----------------------

## Returns

true when the object is touching a wall or any other hitable object

## See Also

[isTouchingWall](#), [isTouchingObject](#)

### 6.2.3.17 bool environment::isTouchingHitableObject ( Location *l*, int *r*, int *id* )

Same as [isTouchingObject\(Location \*l\*, int \*r\*\)](#), but ignores the object with the given id and non-hitable objects.

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object
<i>id</i>	The id of the object to ignore

### 6.2.3.18 bool environment::isTouchingHitableObject ( Location *l*, int *r* )

Checks if the object specified by the given [Location](#) and radius is touching any other hitable objects

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object

## Returns

true when the object is touching any other hitable object

### 6.2.3.19 bool environment::isTouchingHitableObject ( int *id* )

Checks if the object specified by the given id is touching any other hitable objects

## Parameters

<i>id</i>	The id of the object
-----------	----------------------

## Returns

true when the object is touching any other hitable object

### 6.2.3.20 bool environment::isTouchingObject ( Location *l*, int *r*, int *id* )

Same as [isTouchingObject\(Location \*l\*, int \*r\*\)](#), but ignores the object with the given id.

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object
<i>id</i>	The id of the object to ignore

**6.2.3.21** bool environment::isTouchingObject ( [Location](#) *l*, int *r* )

Checks if the object specified by the given [Location](#) and radius is touching any other objects

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object

## Returns

true when the object is touching any other object

**6.2.3.22** bool environment::isTouchingObject ( int *id* )

Checks if the object specified by the given id is touching any other objects

## Parameters

<i>id</i>	The id of the object
-----------	----------------------

## Returns

true when the object is touching any other object

**6.2.3.23** bool environment::isTouchingWall ( [Location](#) *l*, int *r* )

Checks if an object is in the designated window

It returns true whenever the following conditions are met:

x position - radius  $\leq$  0

y position - radius  $\leq$  0

x position + radius  $\geq$  width

y position + radius  $\geq$  height

## Parameters

<i>l</i>	The <a href="#">Location</a> of the object
<i>r</i>	The radius of the object

## Returns

true when the object is touching a wall

**6.2.3.24** bool environment::isTouchingWall ( int *id* )

Checks if an object is in the designated window

It reads in the width, height, x and y position of the object, and then returns true whenever the following conditions are met:

x position - radius  $\leq$  0

y position - radius  $\leq$  0

x position + radius  $\geq$  width

y position + radius  $\geq$  height

#### Parameters

<i>id</i>	The id of the object
-----------	----------------------

#### Returns

true when the object is touching a wall

#### 6.2.3.25 void environment::removeObject ( int *id* )

Removes an object from the environment.

#### Parameters

<i>id</i>	The id of the object to remove
-----------	--------------------------------

## 6.3 util Namespace Reference

util namespace, contains helper functions to add and remove robots

### Functions

- void [reset](#) ()  
*Removes all objects and resets to the initial state.*
- void [display](#) ()  
*Function to render all the objects. This function is called repeatedly from the simulation, and iterates through the objects and calls their respective display functions.*
- void [advance](#) ()  
*Function to update the positions of all objects. This function is called repeatedly from the simulation, and iterates through the objects and calls their respective updatePosition functions.*
- [Color](#) [newColor](#) ()  
*Gets a new, unused color.*
- int [getNumRobotsTargets](#) ()  
*Gets the number of robots/target pairs.*
- int [getNumLights](#) ()  
*Gets the number of lights.*
- int [getNumObstacles](#) ()  
*Gets the number of obstacles.*
- bool [addRobotTarget](#) (int robotType, int lightSensorConnectionPattern, int robotSensorConnectionPattern, int obstacleSensorConnectionPattern, int targetSensorConnectionPattern, float lightSensorScale, float robotSensorScale, float obstacleSensorScale, float targetSensorScale, int initialSpeed, std::string neuralNetworkFile)  
*Adds a robot and paired target to the simulation.*

- bool `addRobot` (int robotType, int lightSensorConnectionPattern, int robotSensorConnectionPattern, int obstacleSensorConnectionPattern, int targetSensorConnectionPattern, float lightSensorScale, float robotSensorScale, float obstacleSensorScale, float targetSensorScale, int initialSpeed, std::string neuralNetworkFile)  
*Adds a robot to the simulation.*
- bool `addNeuralNetworkRobotTarget` (const `NeuralNetwork` &network)  
*Adds a neural network robot and a paired target to the simulation.*
- bool `addStationaryLightSource` ()  
*Adds a light to the simulation.*
- bool `addMovingLightSource` ()  
*Adds a light to the simulation.*
- bool `addObstacle` ()  
*Adds a obstacle to the simulation.*
- bool `copy` (int id, `Location` loc)  
*Copies an object to a new `Location`.*
- bool `removeRobotTarget` ()  
*Removes a robot from the simulation.*
- bool `removeLightSource` ()  
*Removes a light from the simulation.*
- bool `removeObstacle` ()  
*Removes a obstacle from the simulation.*
- void `removeAllRobotTarget` ()  
*Removes all robots.*
- void `removeAllLightSource` ()  
*Removes all lights.*
- void `removeAllObstacle` ()  
*Removes all obstacles.*
- bool `open` (std::string filename)  
*Loads a simulation from a file.*
- bool `save` (std::string filename)  
*Saves the current state to a file.*

### 6.3.1 Detailed Description

util namespace, contains helper functions to add and remove robots

### 6.3.2 Function Documentation

#### 6.3.2.1 bool util::addMovingLightSource ( )

Adds a light to the simulation.

##### Returns

true if successful

#### 6.3.2.2 bool util::addNeuralNetworkRobotTarget ( const `NeuralNetwork` & *network* )

Adds a neural network robot and a paired target to the simulation.

## Parameters

<i>network</i>	the network to control the robot
----------------	----------------------------------

## Returns

true if successful

## 6.3.2.3 bool util::addObstacle ( )

Adds a obstacle to the simulation.

## Returns

true if successful

6.3.2.4 bool util::addRobot ( int *robotType*, int *lightSensorConnectionPattern*, int *robotSensorConnectionPattern*, int *obstacleSensorConnectionPattern*, int *targetSensorConnectionPattern*, float *lightSensorScale*, float *robotSensorScale*, float *obstacleSensorScale*, float *targetSensorScale*, int *initialSpeed*, std::string *neuralNetworkFile* )

Adds a robot to the simulation.

## Parameters

<i>robotType</i>	The type of robot to add. 0 is <a href="#">SimpleRobot</a> , 1 is <a href="#">ComplexRobot</a> , and 2 is <a href="#">NeuralNetworkRobot</a>
<i>lightSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> light sensor
<i>robotSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> robot sensor
<i>obstacleSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> obstacle sensor
<i>targetSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> target sensor
<i>lightSensorScale</i>	Settings for <a href="#">ComplexRobot</a> light sensor
<i>robotSensorScale</i>	Settings for <a href="#">ComplexRobot</a> robot sensor
<i>obstacleSensorScale</i>	Settings for <a href="#">ComplexRobot</a> obstacle sensor
<i>targetSensorScale</i>	Settings for <a href="#">ComplexRobot</a> target sensor
<i>initalSpeed</i>	Default speed for <a href="#">ComplexRobot</a>
<i>neuralNetwork-File</i>	File in which neural network is stored

## Returns

true if successful

6.3.2.5 bool util::addRobotTarget ( int *robotType*, int *lightSensorConnectionPattern*, int *robotSensorConnectionPattern*, int *obstacleSensorConnectionPattern*, int *targetSensorConnectionPattern*, float *lightSensorScale*, float *robotSensorScale*, float *obstacleSensorScale*, float *targetSensorScale*, int *initialSpeed*, std::string *neuralNetworkFile* )

Adds a robot and paired target to the simulation.



## Parameters

<i>robotType</i>	The type of robot to add. 0 is <a href="#">SimpleRobot</a> , 1 is <a href="#">ComplexRobot</a> , and 2 is <a href="#">NeuralNetworkRobot</a>
<i>lightSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> light sensor
<i>robotSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> robot sensor
<i>obstacleSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> obstacle sensor
<i>targetSensor-Connection-Pattern</i>	Settings for <a href="#">ComplexRobot</a> target sensor
<i>lightSensorScale</i>	Settings for <a href="#">ComplexRobot</a> light sensor
<i>robotSensorScale</i>	Settings for <a href="#">ComplexRobot</a> robot sensor
<i>obstacleSensorScale</i>	Settings for <a href="#">ComplexRobot</a> obstacle sensor
<i>targetSensorScale</i>	Settings for <a href="#">ComplexRobot</a> target sensor
<i>initalSpeed</i>	Default speed for <a href="#">ComplexRobot</a>
<i>neuralNetwork-File</i>	File in which neural network is stored

## Returns

true if successful

## 6.3.2.6 bool util::addStationaryLightSource ( )

Adds a light to the simulation.

## Returns

true if successful

## 6.3.2.7 void util::advance ( )

Function to update the positions of all objects. This function is called repeatedly from the simulation, and iterates through the objects and calls their respective updatePosition functions.

## 6.3.2.8 bool util::copy ( int id, Location loc )

Copies an object to a new [Location](#).

## Parameters

<i>id</i>	the id of the object to copy
<i>loc</i>	the destination

## Returns

true if successful

**6.3.2.9 void util::display ( )**

Function to render all the objects. This function is called repeatedly from the simulation, and iterates through the objects and calls their respective display functions.

**6.3.2.10 int util::getNumLights ( )**

Gets the number of lights.

**Returns**

The number of lights

**6.3.2.11 int util::getNumObstacles ( )**

Gets the number of obstacles.

**Returns**

The number of obstacles

**6.3.2.12 int util::getNumRobotsTargets ( )**

Gets the number of robots/target pairs.

**Returns**

The number of robots/target pairs

**6.3.2.13 Color util::newColor ( )**

Gets a new, unused color.

**Returns**

The new color

**6.3.2.14 bool util::open ( std::string *filename* )**

Loads a simulation from a file.

**6.3.2.15 void util::removeAllLightSource ( )**

Removes all lights.

**6.3.2.16 void util::removeAllObstacle ( )**

Removes all obstacles.

**6.3.2.17 void util::removeAllRobotTarget ( )**

Removes all robots.

**6.3.2.18 bool util::removeLightSource ( )**

Removes a light from the simulation.

**Returns**

true if successful

**6.3.2.19 bool util::removeObstacle ( )**

Removes a obstacle from the simulation.

**Returns**

true if successful

**6.3.2.20 bool util::removeRobotTarget ( )**

Removes a robot from the simulation.

**Returns**

true if successful

**6.3.2.21 void util::reset ( )**

Removes all objects and resets to the initial state.

**6.3.2.22 bool util::save ( std::string *filename* )**

Saves the current state to a file.



## Chapter 7

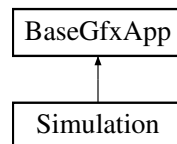
# Class Documentation

### 7.1 BaseGfxApp Class Reference

Graphics driver class.

```
#include <BaseGfxApp.h>
```

Inheritance diagram for BaseGfxApp:



#### Public Member Functions

- [BaseGfxApp](#) (int argc, char \*argv[], int [width](#), int [height](#), int x, int y, int glutFlags, bool createGLUIWin, int gluiWinX, int gluiWinY)
- virtual [~BaseGfxApp](#) ()
- void [setCaption](#) (const std::string &caption)
- void [runMainLoop](#) ()
- virtual void [display](#) ()
- virtual void [advance](#) ()
- virtual void [mouseMoved](#) (int x, int y)
- virtual void [mouseDragged](#) (int x, int y)
- virtual void [leftMouseDown](#) (int x, int y)
- virtual void [leftMouseUp](#) (int x, int y)
- virtual void [rightMouseDown](#) (int x, int y)
- virtual void [rightMouseUp](#) (int x, int y)
- virtual void [middleMouseDown](#) (int x, int y)
- virtual void [middleMouseUp](#) (int x, int y)
- virtual void [keyboard](#) (unsigned char c, int x, int y)
- virtual void [keyboardSpecial](#) (int key, int x, int y)
- virtual void [keyboardUp](#) (unsigned char c, int x, int y)
- virtual void [keyboardSpecialUp](#) (int key, int x, int y)
- virtual void [reshape](#) (int [width](#), int [height](#))
- virtual void [gluiControl](#) (int controlId)
- int [width](#) () const
- int [height](#) () const
- int [handle](#) ()
- GLUI \* [glui](#) ()

## Static Public Member Functions

- static void [graphicsBegin](#) (int v, int delay)

## Static Protected Member Functions

- static void [graphicsTimer](#) (int v)
- static void [s\\_reshape](#) (int [width](#), int [height](#))
- static void [s\\_keyboard](#) (unsigned char c, int x, int y)
- static void [s\\_keyboardspecial](#) (int key, int x, int y)
- static void [s\\_keyboardup](#) (unsigned char c, int x, int y)
- static void [s\\_keyboardspecialup](#) (int key, int x, int y)
- static void [s\\_mousemotion](#) (int x, int y)
- static void [s\\_mousebtn](#) (int b, int s, int x, int y)
- static void [s\\_draw](#) ()
- static void [s\\_glucallback](#) (int controlId)

## Protected Attributes

- int [m\\_glutWindowHandle](#)
- GLUT \* [m\\_glui](#)
- bool [m\\_drag](#)
- int [m\\_width](#)
- int [m\\_height](#)

## Static Protected Attributes

- static [BaseGfxApp](#) \* [s\\_currentApp](#) = NULL
- static bool [s\\_glutInitialized](#) = false

### 7.1.1 Detailed Description

Graphics driver class.

### 7.1.2 Constructor & Destructor Documentation

**7.1.2.1** [BaseGfxApp::BaseGfxApp](#) ( int *argc*, char \* *argv*[], int *width*, int *height*, int *x*, int *y*, int *glutFlags*, bool *createGLUIWin*, int *gluiWinX*, int *gluiWinY* )

**7.1.2.2** [BaseGfxApp::~~BaseGfxApp](#) ( ) [virtual]

### 7.1.3 Member Function Documentation

**7.1.3.1** virtual void [BaseGfxApp::advance](#) ( ) [inline],[virtual]

Reimplemented in [Simulation](#).

**7.1.3.2** virtual void [BaseGfxApp::display](#) ( ) [inline],[virtual]

Reimplemented in [Simulation](#).

7.1.3.3 `GLUI* BaseGfxApp::glui ( ) [inline]`

7.1.3.4 `virtual void BaseGfxApp::gluiControl ( int controlID ) [inline],[virtual]`

Reimplemented in [Simulation](#).

7.1.3.5 `void BaseGfxApp::graphicsBegin ( int v, int delay ) [static]`

This function starts the graphics after a delay.

7.1.3.6 `void BaseGfxApp::graphicsTimer ( int v ) [static],[protected]`

This function calls itself using `glutTimerFunc` once per frame.

7.1.3.7 `int BaseGfxApp::handle ( ) [inline]`

7.1.3.8 `int BaseGfxApp::height ( ) const`

7.1.3.9 `virtual void BaseGfxApp::keyboard ( unsigned char c, int x, int y ) [inline],[virtual]`

Reimplemented in [Simulation](#).

7.1.3.10 `virtual void BaseGfxApp::keyboardSpecial ( int key, int x, int y ) [inline],[virtual]`

Reimplemented in [Simulation](#).

7.1.3.11 `virtual void BaseGfxApp::keyboardSpecialUp ( int key, int x, int y ) [inline],[virtual]`

7.1.3.12 `virtual void BaseGfxApp::keyboardUp ( unsigned char c, int x, int y ) [inline],[virtual]`

7.1.3.13 `virtual void BaseGfxApp::leftMouseDown ( int x, int y ) [inline],[virtual]`

Reimplemented in [Simulation](#).

7.1.3.14 `virtual void BaseGfxApp::leftMouseUp ( int x, int y ) [inline],[virtual]`

Reimplemented in [Simulation](#).

7.1.3.15 `virtual void BaseGfxApp::middleMouseDown ( int x, int y ) [inline],[virtual]`

Reimplemented in [Simulation](#).

7.1.3.16 `virtual void BaseGfxApp::middleMouseUp ( int x, int y ) [inline],[virtual]`

7.1.3.17 `virtual void BaseGfxApp::mouseDragged ( int x, int y ) [inline],[virtual]`

Reimplemented in [Simulation](#).

- 7.1.3.18 `virtual void BaseGfxApp::mouseMoved ( int x, int y )` `[inline], [virtual]`
- 7.1.3.19 `void BaseGfxApp::reshape ( int width, int height )` `[virtual]`
- 7.1.3.20 `virtual void BaseGfxApp::rightMouseDown ( int x, int y )` `[inline], [virtual]`
- 7.1.3.21 `virtual void BaseGfxApp::rightMouseUp ( int x, int y )` `[inline], [virtual]`
- 7.1.3.22 `void BaseGfxApp::runMainLoop ( )`
- 7.1.3.23 `void BaseGfxApp::s_draw ( )` `[static], [protected]`
- 7.1.3.24 `void BaseGfxApp::s_gluicallback ( int controlId )` `[static], [protected]`
- 7.1.3.25 `void BaseGfxApp::s_keyboard ( unsigned char c, int x, int y )` `[static], [protected]`
- 7.1.3.26 `void BaseGfxApp::s_keyboardspecial ( int key, int x, int y )` `[static], [protected]`
- 7.1.3.27 `void BaseGfxApp::s_keyboardspecialup ( int key, int x, int y )` `[static], [protected]`
- 7.1.3.28 `void BaseGfxApp::s_keyboardup ( unsigned char c, int x, int y )` `[static], [protected]`
- 7.1.3.29 `void BaseGfxApp::s_mousebtn ( int b, int s, int x, int y )` `[static], [protected]`
- 7.1.3.30 `void BaseGfxApp::s_mousemotion ( int x, int y )` `[static], [protected]`
- 7.1.3.31 `void BaseGfxApp::s_reshape ( int width, int height )` `[static], [protected]`
- 7.1.3.32 `void BaseGfxApp::setCaption ( const std::string &caption )`
- 7.1.3.33 `int BaseGfxApp::width ( ) const`

#### 7.1.4 Member Data Documentation

- 7.1.4.1 `bool BaseGfxApp::m_drag` `[protected]`
- 7.1.4.2 `GLUI* BaseGfxApp::m_glui` `[protected]`
- 7.1.4.3 `int BaseGfxApp::m_glutWindowHandle` `[protected]`

Underlying glut window handle

- 7.1.4.4 `int BaseGfxApp::m_height` `[protected]`
- 7.1.4.5 `int BaseGfxApp::m_width` `[protected]`
- 7.1.4.6 `BaseGfxApp * BaseGfxApp::s_currentApp = NULL` `[static], [protected]`

GLUT and GLUI event callbacks are sent to the current window/app. Right now, there is only one window anyway (not counting the GLUI UI window.. in the future could be extended to support more windows. In any case, some structure like this is always needed when using glut with C++, since the glut callbacks must be either global or static functions.



7.1.4.7 `bool BaseGfxApp::s_glutInitialized = false` `[static], [protected]`

Has glutInit been called? (only allowed once per program)

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

- [BaseGfxApp.h](#)
- [BaseGfxApp.cpp](#)

## 7.2 Color Struct Reference

This struct holds the representation of a color.

```
#include <Color.h>
```

### Public Member Functions

- [Color](#) ()
- [Color](#) (float r, float g, float b)
- [Color](#) (char c)  
*Creates a RGB color from its char color name The following is the list of allowed colors:*
- [Color](#) (int c)  
*Creates an RGB color from a hex value The color is stored as the last 3 bytes, as 0x<unused (set to 0)><red><green><blue>  
Each byte is interpreted as an integer holding an RGB value from 0-255. These are then converted to floats between 0 and 1 by dividing by 255. Note that this requires at least 3 byte integers.*
- bool [isSimilar](#) ([Color](#) c1)  
*Checks if two colors are similar.*
- std::ostream & [operator<<](#) (std::ostream &out)
- bool [operator==](#) ([Color](#) c1)  
*Checks if two colors are equal.*
- bool [operator!=](#) ([Color](#) c1)  
*Checks if two colors are not equal.*

### Public Attributes

- float [red](#)
- float [green](#)
- float [blue](#)

#### 7.2.1 Detailed Description

This struct holds the representation of a color.

Each field is a float value that should be between 0 and 1. This allows a direct translation from hex colors, with each field being the hex RGB value / 255.

#### 7.2.2 Constructor & Destructor Documentation

7.2.2.1 `Color::Color ( )` `[inline]`

7.2.2.2 `Color::Color ( float r, float g, float b )` `[inline]`

### 7.2.2.3 Color::Color ( char c )

Creates a RGB color from its char color name The following is the list of allowed colors:

Author

Lucas Kramer

Color char	Color name	RGB floating-point value
'R'	Red	(1, 0, 0)
'O'	Orange	(1, 0.4, 0)
'Y'	Yellow	(1, 1, 0)
'G'	Green	(0, 1, 0)
'B'	Blue	(0, 0, 1)
'V'	Violet	(0.5, 0, 0.5)
'W'	White	(1, 1, 1)
default	Black	(0, 0, 0)

### 7.2.2.4 Color::Color ( int c )

Creates an RGB color from a hex value The color is stored as the last 3 bytes, as 0x<unused>(set to 0)><red><green><blue>

Each byte is interpreted as an integer holding an RGB value from 0-255. These are then converted to floats between 0 and 1 by dividing by 255. Note that this requires at least 3 byte integers.

Author

Carl Bahn Lucas Kramer

## 7.2.3 Member Function Documentation

### 7.2.3.1 bool Color::isSimilar ( Color c1 )

Checks if two colors are similar.

Author

Lucas Kramer

Parameters

<i>c1</i>	the color to compare
-----------	----------------------

Returns

true if the colors are similar

### 7.2.3.2 bool Color::operator!=( Color c1 )

Checks if two colors are not equal.

Author

Lucas Kramer

## Parameters

<i>c1</i>	the color to check
-----------	--------------------

## Returns

true if the colors are not equal

7.2.3.3 `std::ostream& Color::operator<< ( std::ostream & out )` `[inline]`

7.2.3.4 `bool Color::operator== ( Color c1 )`

Checks if two colors are equal.

## Author

Lucas Kramer

## Parameters

<i>c1</i>	the color to check
-----------	--------------------

## Returns

true if the colors are equal

## 7.2.4 Member Data Documentation

7.2.4.1 `float Color::blue`

7.2.4.2 `float Color::green`

7.2.4.3 `float Color::red`

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

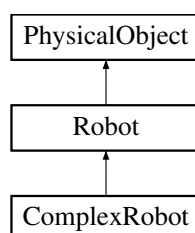
- [Color.h](#)
- [Color.cpp](#)

## 7.3 ComplexRobot Class Reference

A complex robot with configurable feedback from all sensors.

```
#include <ComplexRobot.h>
```

Inheritance diagram for ComplexRobot:



## Public Member Functions

- [ComplexRobot](#) ()
- [ComplexRobot](#) (int radius, [Color](#) color, [Color](#) lineColor, bool [enableLightSensors](#), bool [enableRobotSensors](#), bool [enableObstacleSensors](#), bool [enableTargetSensors](#), bool [lightSensorsCrossed](#), bool [robotSensorsCrossed](#), bool [obstacleSensorsCrossed](#), bool [targetSensorsCrossed](#), float [lightSensorScale](#), float [robotSensorScale](#), float [obstacleSensorScale](#), float [targetSensorScale](#), int [defaultSpeed](#), int targetId=-1)
- [ComplexRobot](#) (int radius, [Location](#) loc, [Color](#) color, [Color](#) lineColor, bool [enableLightSensors](#), bool [enableRobotSensors](#), bool [enableObstacleSensors](#), bool [enableTargetSensors](#), bool [lightSensorsCrossed](#), bool [robotSensorsCrossed](#), bool [obstacleSensorsCrossed](#), bool [targetSensorsCrossed](#), float [lightSensorScale](#), float [robotSensorScale](#), float [obstacleSensorScale](#), float [targetSensorScale](#), int [defaultSpeed](#), int targetId=-1)
- [~ComplexRobot](#) ()
- float [getLeftSpeed](#) (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)
- float [getRightSpeed](#) (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)

## Public Attributes

- const bool [enableLightSensors](#)
- const bool [enableRobotSensors](#)
- const bool [enableObstacleSensors](#)
- const bool [enableTargetSensors](#)
- const bool [lightSensorsCrossed](#)
- const bool [robotSensorsCrossed](#)
- const bool [obstacleSensorsCrossed](#)
- const bool [targetSensorsCrossed](#)
- const float [lightSensorScale](#)
- const float [robotSensorScale](#)
- const float [obstacleSensorScale](#)
- const float [targetSensorScale](#)
- const int [defaultSpeed](#)

## Additional Inherited Members

### 7.3.1 Detailed Description

A complex robot with configurable feedback from all sensors.

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 [ComplexRobot::ComplexRobot](#) ( )

**7.3.2.2** [ComplexRobot::ComplexRobot](#) ( int radius, [Color](#) color, [Color](#) lineColor, bool [enableLightSensors](#), bool [enableRobotSensors](#), bool [enableObstacleSensors](#), bool [enableTargetSensors](#), bool [lightSensorsCrossed](#), bool [robotSensorsCrossed](#), bool [obstacleSensorsCrossed](#), bool [targetSensorsCrossed](#), float [lightSensorScale](#), float [robotSensorScale](#), float [obstacleSensorScale](#), float [targetSensorScale](#), int [defaultSpeed](#), int targetId = -1 )

**7.3.2.3** [ComplexRobot::ComplexRobot](#) ( int radius, [Location](#) loc, [Color](#) color, [Color](#) lineColor, bool [enableLightSensors](#), bool [enableRobotSensors](#), bool [enableObstacleSensors](#), bool [enableTargetSensors](#), bool [lightSensorsCrossed](#), bool [robotSensorsCrossed](#), bool [obstacleSensorsCrossed](#), bool [targetSensorsCrossed](#), float [lightSensorScale](#), float [robotSensorScale](#), float [obstacleSensorScale](#), float [targetSensorScale](#), int [defaultSpeed](#), int targetId = -1 )

7.3.2.4 `ComplexRobot::~~ComplexRobot ( )`

### 7.3.3 Member Function Documentation

7.3.3.1 `float ComplexRobot::getLeftSpeed ( float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal )` `[virtual]`

Implements [Robot](#).

7.3.3.2 `float ComplexRobot::getRightSpeed ( float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal )` `[virtual]`

Implements [Robot](#).

### 7.3.4 Member Data Documentation

7.3.4.1 `const int ComplexRobot::defaultSpeed`

7.3.4.2 `const bool ComplexRobot::enableLightSensors`

7.3.4.3 `const bool ComplexRobot::enableObstacleSensors`

7.3.4.4 `const bool ComplexRobot::enableRobotSensors`

7.3.4.5 `const bool ComplexRobot::enableTargetSensors`

7.3.4.6 `const float ComplexRobot::lightSensorScale`

7.3.4.7 `const bool ComplexRobot::lightSensorsCrossed`

7.3.4.8 `const float ComplexRobot::obstacleSensorScale`

7.3.4.9 `const bool ComplexRobot::obstacleSensorsCrossed`

7.3.4.10 `const float ComplexRobot::robotSensorScale`

7.3.4.11 `const bool ComplexRobot::robotSensorsCrossed`

7.3.4.12 `const float ComplexRobot::targetSensorScale`

7.3.4.13 `const bool ComplexRobot::targetSensorsCrossed`

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

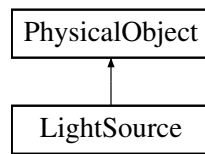
- [ComplexRobot.h](#)
- [ComplexRobot.cpp](#)

## 7.4 LightSource Class Reference

[LightSource](#) for the robot to seek.

```
#include <LightSource.h>
```

Inheritance diagram for LightSource:



## Public Member Functions

- [LightSource](#) (int radius, [Color](#) color)
- [LightSource](#) (int maxRadius, int minRadius, [Color](#) color)
- [LightSource](#) (int radius, [Location](#) loc, [Color](#) color)
- bool [handleCollision](#) (int otherId, bool wasHit)
- [~LightSource](#) ()
- void [display](#) ()

## Additional Inherited Members

### 7.4.1 Detailed Description

[LightSource](#) for the robot to seek.

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 [LightSource::LightSource](#) ( int *radius*, [Color](#) *color* )

This constructor requires two parameters: radius and color)

##### Parameters

<i>radius</i>	the radius in pixels
<i>color</i>	a struct of float value 0 to 1 representing a hexadecimal color

##### Initialization

[LightSource](#) is initialized with the following values:

Orientation: random

Speed: 0

Position: (loc.x, loc.y)

#### 7.4.2.2 [LightSource::LightSource](#) ( int *maxRadius*, int *minRadius*, [Color](#) *color* )

This constructor creates a light with a random radius

##### Parameters

<i>minRadius</i>	the minimum possible radius in pixels
<i>maxRadius</i>	the maximum possible radius in pixels

<i>color</i>	a struct representing color
--------------	-----------------------------

#### Initialization

[LightSource](#) is initialized with the following values:

Orientation: random

Speed: 0

Position: (loc.x, loc.y)

#### 7.4.2.3 LightSource::LightSource ( int *radius*, Location *loc*, Color *color* )

This constructor requires four parameters: radius, color, start [Location](#) and ConfigTable value)

##### Parameters

<i>radius</i>	the radius in pixels
<i>color</i>	a struct of float value 0 to 1 representing a hexadecimal color
<i>loc</i>	a struct of the x,y start <a href="#">Location</a> in pixels
<i>config</i>	a configuration table

#### Initialization

[LightSource](#) is initialized with the following values:

Orientation: 0

Speed: 0

Position: (loc.x, loc.y)

#### 7.4.2.4 LightSource::~~LightSource ( )

This is the class destructor.

### 7.4.3 Member Function Documentation

#### 7.4.3.1 void LightSource::display ( ) [virtual]

This function displays the [LightSource](#) in the graphics.

Reimplemented from [PhysicalObject](#).

#### 7.4.3.2 bool LightSource::handleCollision ( int *otherId*, bool *wasHit* ) [virtual]

##### Author

Lucas Kramer This function gives [LightSource](#) an appropriate reaction when a collision occurs.

##### Parameters

<i>otherId</i>	id of the object that it collides with
----------------	--

<i>wasHit</i>	tells if the object was hit previously
---------------	--

Implements [PhysicalObject](#).

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

- [LightSource.h](#)
- [LightSource.cpp](#)

## 7.5 Location Struct Reference

an xy position on the screen, approximately in pixels x is horizontal from the left and y is vertical from the bottom

```
#include <Location.h>
```

### Public Member Functions

- [Location](#) ()
- [Location](#) (float *x*, float *y*)
- bool [operator==](#) ([Location](#) other)

### Public Attributes

- float *x*
- float *y*

#### 7.5.1 Detailed Description

an xy position on the screen, approximately in pixels x is horizontal from the left and y is vertical from the bottom

#### 7.5.2 Constructor & Destructor Documentation

7.5.2.1 [Location::Location](#) ( ) `[inline]`

7.5.2.2 [Location::Location](#) ( float *x*, float *y* ) `[inline]`

#### 7.5.3 Member Function Documentation

7.5.3.1 [bool Location::operator==](#) ( [Location](#) *other* ) `[inline]`

#### 7.5.4 Member Data Documentation

7.5.4.1 float [Location::x](#)

7.5.4.2 float [Location::y](#)

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

- [Location.h](#)



## 7.6 NeuralNetwork Class Reference

A representation of a neural network.

```
#include <NeuralNetwork.h>
```

### Public Member Functions

- [NeuralNetwork](#) (const [NeuralNetwork](#) &other)
- [NeuralNetwork](#) (const std::string &filename)
- std::vector< float > [compute](#) (const std::vector< float > &inputs)
- [NeuralNetwork mutate](#) (int numChanged, float amount)
- [NeuralNetwork combine](#) (const [NeuralNetwork](#) &other)
- void [write](#) (const std::string &filename)

### Static Public Member Functions

- static [NeuralNetwork load](#) (const std::string &filename)

#### 7.6.1 Detailed Description

A representation of a neural network.

#### 7.6.2 Constructor & Destructor Documentation

##### 7.6.2.1 NeuralNetwork::NeuralNetwork ( const NeuralNetwork & other )

The copy constructor

Parameters

<i>other</i>	the network to copy
--------------	---------------------

##### 7.6.2.2 NeuralNetwork::NeuralNetwork ( const std::string & filename )

Constructs a network from a file description

Parameters

<i>filename</i>	the description filename
-----------------	--------------------------

#### 7.6.3 Member Function Documentation

##### 7.6.3.1 NeuralNetwork NeuralNetwork::combine ( const NeuralNetwork & other )

Combines the network with another network by constructing a new network with half the connection strengths from each. Throws an exception if the networks have incompatible structures

Returns

the new network

7.6.3.2 `vector< float > NeuralNetwork::compute ( const std::vector< float > & inputs )`

Computes the output values of a network for a vector of inputs. Throws an exception if an incorrect number of inputs is provided

## Parameters

<i>inputs</i>	a vector containing the inputs
---------------	--------------------------------

## Returns

a vector containing the outputs

### 7.6.3.3 NeuralNetwork NeuralNetwork::load ( const std::string & filename ) [static]

Loads a network from a file

## Parameters

<i>filename</i>	the filename to load
-----------------	----------------------

## Returns

the new network

### 7.6.3.4 NeuralNetwork NeuralNetwork::mutate ( int numChanged, float amount )

Mutates the network by changing numChanged connection strengths by a random number between -amount and amount

## Parameters

<i>numChanged</i>	the number of connections to mutate
<i>amount</i>	the maximum amount to change the connections

## Returns

the new network

### 7.6.3.5 void NeuralNetwork::write ( const std::string & filename )

Writes the network to a file

## Parameters

<i>filename</i>	the file to write
-----------------	-------------------

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

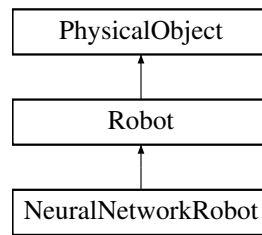
- [NeuralNetwork.h](#)
- [NeuralNetwork.cpp](#)

## 7.7 NeuralNetworkRobot Class Reference

A robot controlled by a neural network with inputs from the sensors.

```
#include <NeuralNetworkRobot.h>
```

Inheritance diagram for NeuralNetworkRobot:



## Public Member Functions

- **NeuralNetworkRobot** (int radius, [Color](#) color, [Color](#) lineColor, [NeuralNetwork](#) network, int targetId=-1, std::string filename=GET\_STRING("DEFAULT\_NEURAL\_NETWORK\_FILE"))
- **NeuralNetworkRobot** (int radius, [Location](#) loc, [Color](#) color, [Color](#) lineColor, [NeuralNetwork](#) network, int targetId=-1, std::string filename=GET\_STRING("DEFAULT\_NEURAL\_NETWORK\_FILE"))
- **~NeuralNetworkRobot** ()
- float **getLeftSpeed** (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)
- float **getRightSpeed** (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)

## Public Attributes

- const std::string [filename](#)

## Additional Inherited Members

### 7.7.1 Detailed Description

A robot controlled by a neural network with inputs from the sensors.

### 7.7.2 Constructor & Destructor Documentation

**7.7.2.1** `NeuralNetworkRobot::NeuralNetworkRobot ( int radius, Color color, Color lineColor, NeuralNetwork network, int targetId = -1, std::string filename = GET_STRING ( "DEFAULT_NEURAL_NETWORK_FILE" ) )`

**7.7.2.2** `NeuralNetworkRobot::NeuralNetworkRobot ( int radius, Location loc, Color color, Color lineColor, NeuralNetwork network, int targetId = -1, std::string filename = GET_STRING ( "DEFAULT_NEURAL_NETWORK_FILE" ) )`

**7.7.2.3** `NeuralNetworkRobot::~~NeuralNetworkRobot ( )`

### 7.7.3 Member Function Documentation

**7.7.3.1** `float NeuralNetworkRobot::getLeftSpeed ( float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal ) [virtual]`

Implements [Robot](#).

7.7.3.2 `float NeuralNetworkRobot::getRightSpeed ( float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal )` `[virtual]`

Implements [Robot](#).

## 7.7.4 Member Data Documentation

7.7.4.1 `const std::string NeuralNetworkRobot::filename`

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

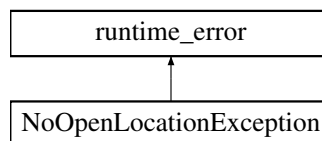
- [NeuralNetworkRobot.h](#)
- [NeuralNetworkRobot.cpp](#)

## 7.8 NoOpenLocationException Class Reference

An exception to be thrown when an open [Location](#) cannot be found.

```
#include <environment.h>
```

Inheritance diagram for NoOpenLocationException:



### Public Member Functions

- [NoOpenLocationException](#) ()

### 7.8.1 Detailed Description

An exception to be thrown when an open [Location](#) cannot be found.

### 7.8.2 Constructor & Destructor Documentation

7.8.2.1 `NoOpenLocationException::NoOpenLocationException ( )` `[inline]`

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

- [environment.h](#)

## 7.9 ObjectIterator Class Reference

[ObjectIterator](#) class, a simple iterator for the object in environment.

```
#include <environment.h>
```

## Public Member Functions

- [ObjectIterator](#) ()  
*The default constructor, the iterated objects are allways the iterObjects vector in environment.*
- [ObjectIterator](#) (const [ObjectIterator](#) &other)  
*The copy constructor, the iterated objects are allways the iterObjects vector in environment.*
- [~ObjectIterator](#) ()  
*The destructor, does nothing.*
- void [operator++](#) ()
- void [operator++](#) (int)
- [PhysicalObject](#) \* [operator\\*](#) ()
- bool [operator>](#) (const [ObjectIterator](#) &other)
- bool [operator<](#) (const [ObjectIterator](#) &other)
- bool [operator==](#) (const [ObjectIterator](#) &other)
- bool [operator!=](#) (const [ObjectIterator](#) &other)

## Friends

- [environment::objectIterator](#) [environment::getObjectsBegin](#) ()
- [environment::objectIterator](#) [environment::getObjectsEnd](#) ()

### 7.9.1 Detailed Description

[ObjectIterator](#) class, a simple iterator for the object in environment.

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 [ObjectIterator::ObjectIterator](#) ( )

The default constructor, the iterated objects are allways the iterObjects vector in environment.

#### 7.9.2.2 [ObjectIterator::ObjectIterator](#) ( const [ObjectIterator](#) & *other* )

The copy constructor, the iterated objects are allways the iterObjects vector in environment.

Parameters

<i>other</i>	The <a href="#">ObjectIterator</a> to copy
--------------	--

#### 7.9.2.3 [ObjectIterator::~~ObjectIterator](#) ( )

The destructor, does nothing.

### 7.9.3 Member Function Documentation

#### 7.9.3.1 bool [ObjectIterator::operator!=](#) ( const [ObjectIterator](#) & *other* )

#### 7.9.3.2 [PhysicalObject](#) \* [ObjectIterator::operator\\*](#) ( )

#### 7.9.3.3 void [ObjectIterator::operator++](#) ( )

7.9.3.4 void ObjectIterator::operator++ ( int )

7.9.3.5 bool ObjectIterator::operator< ( const ObjectIterator & other )

7.9.3.6 bool ObjectIterator::operator== ( const ObjectIterator & other )

7.9.3.7 bool ObjectIterator::operator> ( const ObjectIterator & other )

## 7.9.4 Friends And Related Function Documentation

7.9.4.1 environment::objectIterator environment::getObjectsBegin ( ) [friend]

7.9.4.2 environment::objectIterator environment::getObjectsEnd ( ) [friend]

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

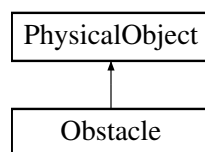
- [environment.h](#)
- [environment.cpp](#)

## 7.10 Obstacle Class Reference

[Obstacle](#) for the robot to hit/avoid.

```
#include <Obstacle.h>
```

Inheritance diagram for Obstacle:



### Public Member Functions

- [Obstacle](#) (int radius, [Color](#) color)
- [Obstacle](#) (int maxRadius, int minRadius, [Color](#) color)
- [Obstacle](#) (int radius, [Location](#) loc, [Color](#) color)
- bool [handleCollision](#) (int otherId, bool wasHit)
- [~Obstacle](#) ()

### Additional Inherited Members

#### 7.10.1 Detailed Description

[Obstacle](#) for the robot to hit/avoid.

#### 7.10.2 Constructor & Destructor Documentation

7.10.2.1 Obstacle::Obstacle ( int *radius*, [Color](#) *color* )

## Parameters

<i>radius</i>	the radius in pixels
<i>color</i>	a struct of float value 0 to 1 representing a hexadecimal color
<i>config</i>	a configuration table

## Initialization

[Obstacle](#) is initialized with the following values:

Radius: DEFAULT\_RADIUS as specified in ../config/default

Orientation: random

Speed: 0

Position: (loc.x, loc.y)

#### 7.10.2.2 Obstacle::Obstacle ( int *maxRadius*, int *minRadius*, Color *color* )

## Parameters

<i>minRadius</i>	the minimum radius in pixels
<i>maxRadius</i>	the maximum radius in pixels
<i>color</i>	a struct of float value 0 to 1 representing a hexadecimal color

## Initialization

[Obstacle](#) is initialized with the following values:

Orientation: random

Speed: 0

Position: (loc.x, loc.y)

#### 7.10.2.3 Obstacle::Obstacle ( int *radius*, Location *loc*, Color *color* )

## Parameters

<i>radius</i>	the radius in pixels
<i>color</i>	a struct of float value 0 to 1 representing a hexadecimal color
<i>loc</i>	a struct of the x,y start <a href="#">Location</a> in pixels

## Initialization

[Obstacle](#) is initialized with the following values:

Orientation: 0

Speed: 0

Position: (loc.x, loc.y)

#### 7.10.2.4 Obstacle::~Obstacle ( )

This is the class destructor.

### 7.10.3 Member Function Documentation

#### 7.10.3.1 bool Obstacle::handleCollision ( int *otherId*, bool *wasHit* ) [virtual]



**Author**

Lucas Kramer This function gives [Obstacle](#) an appropriate reaction when a collision occurs. Does nothing for now

**Parameters**

<i>otherId</i>	id of the object that it collides with
<i>wasHit</i>	tells if the object was hit previously

Implements [PhysicalObject](#).

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

- [Obstacle.h](#)
- [Obstacle.cpp](#)

## 7.11 OptimizeSimulation Class Reference

[OptimizeSimulation](#) class, sets up environments and robots.

```
#include <OptimizeSimulation.h>
```

**Public Member Functions**

- [OptimizeSimulation](#) (int argc, char \*argv[])  
The constructor for the [Simulation](#) class.
- virtual [~OptimizeSimulation](#) ()
- void [runMainLoop](#) ()
- void [stop](#) ()
- int [getPerformance](#) (const [NeuralNetwork](#) &network)
- int [getPerformanceRandomRepeated](#) (const [NeuralNetwork](#) &network)
- int [getPerformanceMaze](#) (const [NeuralNetwork](#) &network)
- int [getPerformanceObstacles](#) (const [NeuralNetwork](#) &network)

### 7.11.1 Detailed Description

[OptimizeSimulation](#) class, sets up environments and robots.

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 [OptimizeSimulation::OptimizeSimulation](#) ( int *argc*, char \* *argv*[] )

The constructor for the [Simulation](#) class.

**Parameters**

<i>argc</i>	The number of command-line arguments
<i>argv</i>	The command-line arguments

#### 7.11.2.2 [OptimizeSimulation::~~OptimizeSimulation](#) ( ) [virtual]

### 7.11.3 Member Function Documentation

- 7.11.3.1 `int OptimizeSimulation::getPerformance ( const NeuralNetwork & network )`
- 7.11.3.2 `int OptimizeSimulation::getPerformanceMaze ( const NeuralNetwork & network )`
- 7.11.3.3 `int OptimizeSimulation::getPerformanceObstacles ( const NeuralNetwork & network )`
- 7.11.3.4 `int OptimizeSimulation::getPerformanceRandomRepeated ( const NeuralNetwork & network )`
- 7.11.3.5 `void OptimizeSimulation::runMainLoop ( )`
- 7.11.3.6 `void OptimizeSimulation::stop ( )`

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

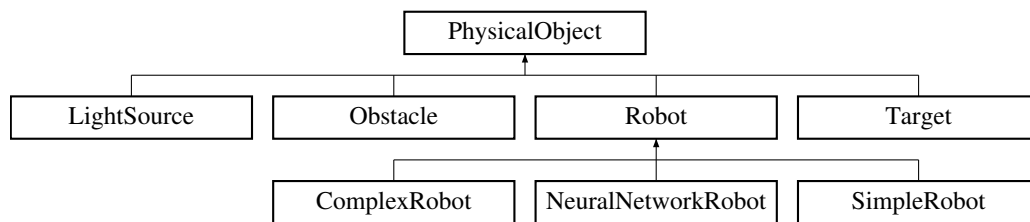
- [OptimizeSimulation.h](#)
- [OptimizeSimulation.cpp](#)

## 7.12 PhysicalObject Class Reference

This is a common superclass for all objects.

```
#include <PhysicalObject.h>
```

Inheritance diagram for PhysicalObject:



### Public Member Functions

- `PhysicalObject (ObjectType objectType, int radius=GET_INT("DEFAULT_RADIUS"), Color color=Color('?'), bool isHitable=true)`
- `PhysicalObject (ObjectType objectType, int maxRadius, int minRadius, Color color=Color('?'), bool isHitable=true)`
- `PhysicalObject (ObjectType objectType, int radius, Location loc, Color color=Color('?'), bool isHitable=true)`
- `virtual ~PhysicalObject ()`
- `int getId () const`
- `void setPosition (float x, float y)`
- `void forceSetPosition (float x, float y)`
- `void setLocation (Location loc)`
- `void reorient (int angle, float distance)`
- `bool translate (float distance)`
- `void forceTranslate (float distance)`
- `void setOrientation (int degrees)`
- `void setSpeed (int pps)`
- `void setRadius (int radius)`
- `void setColor (Color color)`
- `float getXPosition () const`
- `float getYPosition () const`

- [Location getLocation](#) () const
- int [getOrientation](#) () const
- int [getSpeed](#) () const
- int [getRadius](#) () const
- [Color getColor](#) () const
- void [pointTo](#) (const [PhysicalObject](#) &other)
- void [rotate](#) (int degrees)
- bool [hasEqualPosition](#) (const [PhysicalObject](#) &other) const
- bool [updatePosition](#) ()
- virtual void [update](#) ()
- virtual void [updateMembers](#) ()
- virtual void [display](#) ()
- virtual bool [handleCollision](#) (int otherId, bool wasHit)=0

### Public Attributes

- const [ObjectType objectType](#)
- const bool [isHitable](#)

### Static Protected Member Functions

- static [Location findOpenLocation](#) (int radius)

#### 7.12.1 Detailed Description

This is a common superclass for all objects.

#### 7.12.2 Constructor & Destructor Documentation

**7.12.2.1** `PhysicalObject::PhysicalObject ( ObjectType objectType, int radius = GET_INT ( "DEFAULT_RADIUS" ), Color color = Color ( ' ? ' ), bool isHitable = true )`

**7.12.2.2** `PhysicalObject::PhysicalObject ( ObjectType objectType, int maxRadius, int minRadius, Color color = Color ( ' ? ' ), bool isHitable = true )`

**7.12.2.3** `PhysicalObject::PhysicalObject ( ObjectType objectType, int radius, Location loc, Color color = Color ( ' ? ' ), bool isHitable = true )`

**7.12.2.4** `PhysicalObject::~~PhysicalObject ( )` [virtual]

This is the class destructor, it is implemented by all subclasses.

#### 7.12.3 Member Function Documentation

**7.12.3.1** `void PhysicalObject::display ( )` [virtual]

This is implemented by each class, normally just calls displayAsCircle.

#### See Also

`displayAsCircle()`

Reimplemented in [Robot](#), and [LightSource](#).

#### 7.12.3.2 Location `PhysicalObject::findOpenLocation ( int radius )` `[static], [protected]`

This function finds an open [Location](#) to place the object

## Parameters

<i>radius</i>	The radius of the object
---------------	--------------------------

7.12.3.3 void PhysicalObject::forceSetPosition ( float *x*, float *y* )

Sets the position for the object. It does not throw an exception on invalid input.

## Parameters

<i>x</i>	position in pixels
<i>y</i>	position in pixels

7.12.3.4 void PhysicalObject::forceTranslate ( float *distance* )

Translates without calling handleCollision. This may cause objects to overlap

## Parameters

<i>distance</i>	Distance to move in pixels
-----------------	----------------------------

## 7.12.3.5 Color PhysicalObject::getColor ( ) const

Returns the color of the object

## Returns

color

## 7.12.3.6 int PhysicalObject::getId ( ) const

Getter function for the object's id.

## Returns

The object's id

## 7.12.3.7 Location PhysicalObject::getLocation ( ) const

Returns the x and y [Location](#) of the object in pixels in a struct

## See Also

[Location](#)

## 7.12.3.8 int PhysicalObject::getOrientation ( ) const

Returns the orientation of the object in degrees

## Returns

orientation in degrees

### 7.12.3.9 `int PhysicalObject::getRadius ( ) const`

Returns the radius of the object

#### Returns

radius

### 7.12.3.10 `int PhysicalObject::getSpeed ( ) const`

Returns the speed of the object

#### Returns

speed in pixels per second

### 7.12.3.11 `float PhysicalObject::getXPosition ( ) const`

Returns the x position in pixels of the object

#### Returns

x position, horizontal from left

### 7.12.3.12 `float PhysicalObject::getYPosition ( ) const`

Returns the y position in pixels of the object

#### Returns

y position, vertical from bottom

### 7.12.3.13 `virtual bool PhysicalObject::handleCollision ( int otherId, bool wasHit ) [pure virtual]`

This is implemented by each class. It controls the objects behavior on collision during translate, normally just calls `reorient`.

#### Parameters

<i>otherId</i>	The id of the object it hit, or -1 for a wall
<i>wasHit</i>	true if the object hit something, false if hit by something (see <code>translate</code> )

#### Returns

true if objects were added or removed

#### See Also

[reorient\(\)](#)

Implemented in [Robot](#), [LightSource](#), [Obstacle](#), and [Target](#).

### 7.12.3.14 `bool PhysicalObject::hasEqualPosition ( const PhysicalObject & other ) const`

This member function checks when two objects are at the same [Location](#), it has one parameter

## Parameters

<i>other</i>	The object to compare
--------------	-----------------------

## Returns

A boolean value that is true when objects are overlapping

7.12.3.15 void PhysicalObject::pointTo ( const PhysicalObject & *other* )

Calculates the direction of the object is facing towards the target object based on position of the two objects using the math function atan2

## Parameters

<i>other</i>	A <a href="#">PhysicalObject</a> class pointer
--------------	--

7.12.3.16 void PhysicalObject::reorient ( int *angle*, float *distance* )

This function can be called by subclasses when handling a collision. This causes the robot to rotate by the specified amount, and then translate if that is enabled in the configuration. Since translating could cause another collision, this could call back to reorient, causing infinite recursion in some cases. To prevent this, the function omits the translate if it is more than MAX\_REORIENT\_RETRIES deep in recursive calls, set in the configuration

## Parameters

<i>angle</i>	Angle to rotate in degrees
<i>distance</i>	Distance to move in pixels

7.12.3.17 void PhysicalObject::rotate ( int *degrees* )

This function increments the object's orientation

## Parameters

<i>degrees</i>	Angle to be incremented clockwise.
----------------	------------------------------------

## Exceptions

<i>Input</i>	must be within (-360,360)
--------------	---------------------------

7.12.3.18 void PhysicalObject::setColor ( Color *color* )

Set a new color for the object.

## Parameters

<i>color</i>	
--------------	--

## See Also

[Color](#)

7.12.3.19 void PhysicalObject::setLocation ( Location *loc* )

Sets the position for the object for the object

## Parameters

<i>loc</i>	A struct of the x,y <a href="#">Location</a> in pixels
------------	--

## See Also

[setPosition](#)

7.12.3.20 void PhysicalObject::setOrientation ( int *degrees* )

Sets the absolute orientation for an object

## Parameters

<i>degrees</i>	New angle for the object in degrees clockwise of North.
----------------	---

7.12.3.21 void PhysicalObject::setPosition ( float *x*, float *y* )

Sets the position for the object

## Parameters

<i>x</i>	x position in pixels
<i>y</i>	y position in pixels

7.12.3.22 void PhysicalObject::setRadius ( int *radius* )

Set the radius of the object

## Parameters

<i>radius</i>	the new radius in pixels
---------------	--------------------------

7.12.3.23 void PhysicalObject::setSpeed ( int *pps* )

Sets the speed of an object

## Parameters

<i>pps</i>	The new speed in pixels per second
------------	------------------------------------

7.12.3.24 bool PhysicalObject::translate ( float *distance* )

Translates in the current direction, and calls handleCollision where needed

## Parameters

<i>distance</i>	Distance to move in pixels
-----------------	----------------------------

## Returns

true if objects were added or removed when any collisions were handled. This is to allow the position update function to break out of the loop when objects are removed, to prevent trying to update objects that no longer exist



#### 7.12.3.25 void PhysicalObject::update ( ) [virtual]

This is implemented by each class, and performs random updates, e.g. point toward target

Reimplemented in [Robot](#).

#### 7.12.3.26 void PhysicalObject::updateMembers ( ) [virtual]

This is implemented by each class, and performs updates such as setting the position of sub-objects

Reimplemented in [Robot](#).

#### 7.12.3.27 bool PhysicalObject::updatePosition ( )

This member function uses the speed and time to update the [Location](#)

Returns

true if objects were added or removed

### 7.12.4 Member Data Documentation

#### 7.12.4.1 const bool PhysicalObject::isHitable

#### 7.12.4.2 const ObjectType PhysicalObject::objectType

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

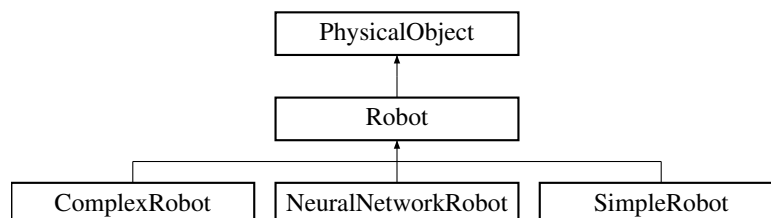
- [PhysicalObject.h](#)
- [PhysicalObject.cpp](#)

## 7.13 Robot Class Reference

[Robot](#) that moves around the window and bumps into obstacles.

```
#include <Robot.h>
```

Inheritance diagram for Robot:



### Public Member Functions

- [Robot](#) ([RobotType](#) robotType, int radius, [Color](#) color, [Color](#) lineColor, int targetId=-1)
- [Robot](#) ([RobotType](#) robotType, int radius, [Location](#) loc, [Color](#) color, [Color](#) lineColor, int targetId=-1)
- [~Robot](#) ( )
- int [getTarget](#) ( ) const
- void [setTarget](#) (int id)

- bool [handleCollision](#) (int otherId, bool wasHit)
- void [update](#) ()
- void [updateMembers](#) ()
- void [display](#) ()
- void [setLineColor](#) ([Color](#) lineColor)
- [Color](#) [getLineColor](#) ()

## Public Attributes

- const [RobotType](#) [robotType](#)

## Protected Member Functions

- virtual float [getLeftSpeed](#) (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)=0
- virtual float [getRightSpeed](#) (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)=0

## Additional Inherited Members

### 7.13.1 Detailed Description

[Robot](#) that moves around the window and bumps into obstacles.

### 7.13.2 Constructor & Destructor Documentation

7.13.2.1 [Robot::Robot](#) ( [RobotType](#) *robotType*, int *radius*, [Color](#) *color*, [Color](#) *lineColor*, int *targetId* = -1 )

#### Parameters

<i>robotType</i>	the robot type
<i>radius</i>	the radius in pixels
<i>color</i>	color of the robot itself, a struct of float value 0 to 1 representing a hexadecimal color
<i>lineColor</i>	color of the line, a struct of float value 0 to 1 representing a hexadecimal color

#### Initialization

[Robot](#) is initialized with the following values:

Orientation: 0

Speed: 0

Position: (loc.x, loc.y)

7.13.2.2 [Robot::Robot](#) ( [RobotType](#) *robotType*, int *radius*, [Location](#) *loc*, [Color](#) *color*, [Color](#) *lineColor*, int *targetId* = -1 )

#### Parameters

<i>robotType</i>	the robot type
<i>radius</i>	the radius in pixels
<i>color</i>	a struct of float value 0 to 1 representing a hexadecimal color
<i>linecolor</i>	color of the line, a struct of float value 0 to 1 representing a hexadecimal color
<i>loc</i>	start <a href="#">Location</a> of the robot

#### Initialization

[Robot](#) is initialized with the following values:

Orientation: random

Speed: 0

Position: (loc.x, loc.y)

#### 7.13.2.3 Robot::~~Robot ( )

This is the class destructor.

### 7.13.3 Member Function Documentation

#### 7.13.3.1 void Robot::display ( ) [virtual]

##### Author

Lucas Kramer This function displays the [Robot](#) in the graphics

Reimplemented from [PhysicalObject](#).

#### 7.13.3.2 virtual float Robot::getLeftSpeed ( float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal ) [protected],[pure virtual]

Implemented in [ComplexRobot](#), [NeuralNetworkRobot](#), and [SimpleRobot](#).

#### 7.13.3.3 Color Robot::getLineColor ( )

#### 7.13.3.4 virtual float Robot::getRightSpeed ( float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal ) [protected],[pure virtual]

Implemented in [ComplexRobot](#), [NeuralNetworkRobot](#), and [SimpleRobot](#).

#### 7.13.3.5 int Robot::getTarget ( ) const

##### Author

Lucas Kramer This function gets the id of the target for the robot to seek.

##### Parameters

---

<i>id</i>	the id of the target
-----------	----------------------

7.13.3.6 `bool Robot::handleCollision ( int otherId, bool wasHit )` [virtual]

#### Author

Lucas Kramer This function gives [Robot](#) an appropriate reaction when a collision occurs

#### Parameters

<i>otherId</i>	id of the object that it collides with
<i>wasHit</i>	tells if the object was hit previously

Implements [PhysicalObject](#).

7.13.3.7 `void Robot::setLineColor ( Color lineColor )`

#### Author

Lucas Kramer This function sets a new color for the robot's direction line.

#### Parameters

<i>lineColor</i>	color of the line, a struct of float value 0 to 1 representing a hexadecimal color
------------------	--

7.13.3.8 `void Robot::setTarget ( int id )`

#### Author

Lucas Kramer This function sets the id of the target for the robot to seek.

#### Returns

the target id

7.13.3.9 `void Robot::update ( )` [virtual]

#### Author

Lucas Kramer This function updates the [Robot](#)'s wheel speeds from the sensors

Reimplemented from [PhysicalObject](#).

7.13.3.10 `void Robot::updateMembers ( )` [virtual]

#### Author

Lucas Kramer This function updates the position of the robot's sub-objects

Reimplemented from [PhysicalObject](#).

### 7.13.4 Member Data Documentation

#### 7.13.4.1 `const RobotType Robot::robotType`

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

- [Robot.h](#)
- [Robot.cpp](#)

## 7.14 Sensor Class Reference

```
#include <Sensor.h>
```

### Public Member Functions

- [Sensor](#) ([Location](#) loc, int orientation, [ObjectType](#) typeDetected)
- [Sensor](#) ([Location](#) loc, int orientation, int viewAngle, [ObjectType](#) typeDetected)
- void [setPosition](#) (float x, float y)
- void [setPosition](#) ([Location](#) loc)
- [Location](#) [getPosition](#) ()
- float [getXPosition](#) ()
- float [getYPosition](#) ()
- void [setOrientation](#) (int orientation)
- int [getOrientation](#) ()
- void [setViewAngle](#) (int degrees)
- int [getViewAngle](#) ()
- void [setTypeDetected](#) ([ObjectType](#) type)
- [ObjectType](#) [getTypeDetected](#) ()
- void [updatePosition](#) ([Location](#) robotLoc, int robotAngle)
- void [display](#) ()
- float [sense](#) ()

### 7.14.1 Constructor & Destructor Documentation

#### 7.14.1.1 `Sensor::Sensor ( Location loc, int orientation, ObjectType typeDetected )`

[Sensor](#) constructor with default angle

Parameters

<i>loc</i>	<a href="#">Location</a> of the sensor in x and y
<i>orientation</i>	orientation of the sensor
<i>typeDetected</i>	type that sensor will detect

#### 7.14.1.2 `Sensor::Sensor ( Location loc, int orientation, int viewAngle, ObjectType typeDetected )`

[Sensor](#) constructor

Parameters

<i>loc</i>	<a href="#">Location</a> of the sensor in x and y
<i>orientation</i>	orientation of the sensor
<i>viewAngle</i>	view angle of the sensor in degrees
<i>typeDetected</i>	type that sensor will detect

## 7.14.2 Member Function Documentation

### 7.14.2.1 void Sensor::display ( )

Displays the sensor

Author

Carl

### 7.14.2.2 int Sensor::getOrientation ( )

Returns the orientation of the sensor

Returns

orientation in degrees

### 7.14.2.3 Location Sensor::getPosition ( )

Returns a [Location](#) struct of x and y position in pixels of the sensor

Returns

A [Location](#) struct of x and y position

### 7.14.2.4 ObjectType Sensor::getTypeDetected ( )

Returns the type that sensor detects

Author

Himawan

Returns

type the sensor detects

### 7.14.2.5 int Sensor::getViewAngle ( )

Returns the view angle of the sensor

Returns

view angle in degrees

#### 7.14.2.6 float Sensor::getXPosition ( )

Returns the x position in pixels of the sensor

##### Returns

x position

#### 7.14.2.7 float Sensor::getYPosition ( )

Returns the y position in pixels of the sensor

##### Returns

y position

#### 7.14.2.8 float Sensor::sense ( )

Returns some strength by sensing the environment

##### Author

Carl & Himawan

##### Returns

the strength of sensor reading of [0..1]

#### 7.14.2.9 void Sensor::setOrientation ( int *orientation* )

Sets the orientation for the sensor

##### Parameters

<i>orientation</i>	The orientation of the object in degrees
--------------------	--

#### 7.14.2.10 void Sensor::setPosition ( float *x*, float *y* )

Sets the position for the sensor

##### Parameters

<i>x</i>	x position in pixels
<i>y</i>	y position in pixels

#### 7.14.2.11 void Sensor::setPosition ( Location *loc* )

Sets the position for the sensor

## Parameters

<i>loc</i>	A struct of the x and y <a href="#">Location</a> in pixels
------------	--

7.14.2.12 void Sensor::setTypeDetected ( **ObjectType** *type* )

Sets the object type that the sensor detects

## Author

Himawan

## Parameters

<i>type</i>	object type to detect
-------------	-----------------------

7.14.2.13 void Sensor::setViewAngle ( int *degrees* )

Sets the view angle of the sensor

## Parameters

<i>degrees</i>	The view angle of the sensor in degrees
----------------	---

7.14.2.14 void Sensor::updatePosition ( **Location** *robotLoc*, int *robotAngle* )

Called to update the position of the sensor match the updated position of the robot

## Author

Carl

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

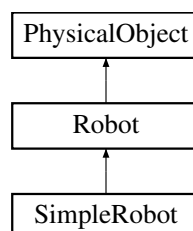
- [Sensor.h](#)
- [Sensor.cpp](#)

## 7.15 SimpleRobot Class Reference

A simple robot with uncrossed feedback.

```
#include <SimpleRobot.h>
```

Inheritance diagram for SimpleRobot:





## Public Member Functions

- [SimpleRobot](#) (int radius, [Color](#) color, [Color](#) lineColor, int targetId=-1)
- [SimpleRobot](#) (int radius, [Location](#) loc, [Color](#) color, [Color](#) lineColor, int targetId=-1)
- [~SimpleRobot](#) ()
- float [getLeftSpeed](#) (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)
- float [getRightSpeed](#) (float leftLightSensorVal, float rightLightSensorVal, float leftRobotSensorVal, float rightRobotSensorVal, float leftObstacleSensorVal, float rightObstacleSensorVal, float leftTargetSensorVal, float rightTargetSensorVal)

## Additional Inherited Members

### 7.15.1 Detailed Description

A simple robot with uncrossed feedback.

### 7.15.2 Constructor & Destructor Documentation

7.15.2.1 [SimpleRobot::SimpleRobot](#) ( int *radius*, [Color](#) *color*, [Color](#) *lineColor*, int *targetId* = -1 )

7.15.2.2 [SimpleRobot::SimpleRobot](#) ( int *radius*, [Location](#) *loc*, [Color](#) *color*, [Color](#) *lineColor*, int *targetId* = -1 )

7.15.2.3 [SimpleRobot::~~SimpleRobot](#) ( )

### 7.15.3 Member Function Documentation

7.15.3.1 float [SimpleRobot::getLeftSpeed](#) ( float *leftLightSensorVal*, float *rightLightSensorVal*, float *leftRobotSensorVal*, float *rightRobotSensorVal*, float *leftObstacleSensorVal*, float *rightObstacleSensorVal*, float *leftTargetSensorVal*, float *rightTargetSensorVal* ) [virtual]

Implements [Robot](#).

7.15.3.2 float [SimpleRobot::getRightSpeed](#) ( float *leftLightSensorVal*, float *rightLightSensorVal*, float *leftRobotSensorVal*, float *rightRobotSensorVal*, float *leftObstacleSensorVal*, float *rightObstacleSensorVal*, float *leftTargetSensorVal*, float *rightTargetSensorVal* ) [virtual]

Implements [Robot](#).

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

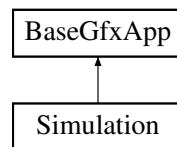
- [SimpleRobot.h](#)
- [SimpleRobot.cpp](#)

## 7.16 Simulation Class Reference

[Simulation](#) class, sets up the GUI and the drawing environment.

```
#include <Simulation.h>
```

Inheritance diagram for [Simulation](#):



## Public Member Functions

- [Simulation](#) (int argc, char \*argv[])  
The constructor for the [Simulation](#) class.
- virtual [~Simulation](#) ()
- void [initObjects](#) ()  
Adds objects according to configuration.
- void [tryAddRobotTarget](#) ()  
Adds a robot and target to the simulation, or raises an error on failure.
- void [tryAddRobot](#) ()  
Adds a robot to the simulation, or raises an error on failure.
- void [tryAddStationaryLightSource](#) ()  
Adds a stationary light to the simulation, or raises an error on failure.
- void [tryAddMovingLightSource](#) ()  
Adds a moving light to the simulation, or raises an error on failure.
- void [tryAddObstacle](#) ()  
Adds a obstacle to the simulation, or raises an error on failure.
- void [tryRemoveRobotTarget](#) ()  
Removes a robot and target from the simulation, or raises an error on failure.
- void [tryRemoveLightSource](#) ()  
Removes a light from the simulation, or raises an error on failure.
- void [tryRemoveObstacle](#) ()  
Removes a obstacle from the simulation, or raises an error on failure.
- void [tryOpen](#) ()  
Loads a simulation, or raises an error on failure.
- void [trySave](#) ()  
Saves a simulation, or raises an error on failure.
- void [display](#) ()  
Call-back function to render all the objects. This function is called repeatedly from glut, and iterates through the objects and calls their respective display functions.
- void [advance](#) ()  
Call-back function to update the positions of all objects. This function is called repeatedly from glut, and iterates through the objects and calls their respective updatePosition functions.
- void [gluiControl](#) (int controlId)  
Call-back function for user input.
- void [leftMouseDown](#) (int x, int y)  
Call-back function for mouse click.
- void [leftMouseUp](#) (int x, int y)  
Call-back function for mouse click.
- void [middleMouseDown](#) (int x, int y)  
Call-back function for mouse click.
- void [mouseDragged](#) (int x, int y)  
Call-back function for mouse drag.
- void [keyboard](#) (unsigned char key, int x, int y)  
Call-back function keyboard press.

- void [keyboardSpecial](#) (int key, int x, int y)  
*Call-back function for special keyboard press.*
- void [start](#) (int)
- void [pause](#) (int)
- void [resume](#) (int)
- void [reset](#) (int)
- void [clear](#) (int)
- void [random](#) (int)

## Static Public Member Functions

- static void [s\\_start](#) (int)
- static void [s\\_pause](#) (int)
- static void [s\\_resume](#) (int)
- static void [s\\_reset](#) (int)
- static void [s\\_clear](#) (int)
- static void [s\\_random](#) (int)
- static void [s\\_addRobotTarget](#) (int)
- static void [s\\_addRobot](#) (int)
- static void [s\\_addStationaryLightSource](#) (int)
- static void [s\\_addMovingLightSource](#) (int)
- static void [s\\_addObstacle](#) (int)
- static void [s\\_removeRobotTarget](#) (int)
- static void [s\\_removeLightSource](#) (int)
- static void [s\\_removeObstacle](#) (int)
- static void [s\\_removeAllRobotTarget](#) (int)
- static void [s\\_removeAllLightSource](#) (int)
- static void [s\\_removeAllObstacle](#) (int)
- static void [s\\_refreshConfiguration](#) (int)
- static void [s\\_open](#) (int)
- static void [s\\_save](#) (int)
- static void [s\\_neuralNetworkFileChanged](#) (int)

## Additional Inherited Members

### 7.16.1 Detailed Description

[Simulation](#) class, sets up the GUI and the drawing environment.

### 7.16.2 Constructor & Destructor Documentation

#### 7.16.2.1 [Simulation::Simulation](#) ( int *argc*, char \* *argv*[] )

The constructor for the [Simulation](#) class.

Parameters

<i>argc</i>	The number of command-line arguments
<i>argv</i>	The command-line arguments

7.16.2.2 `Simulation::~Simulation ( )` [virtual]

### 7.16.3 Member Function Documentation

7.16.3.1 `void Simulation::advance ( )` [virtual]

Call-back function to update the positions of all objects. This function is called repeatedly from glut, and iterates through the objects and calls their respective `updatePosition` functions.

Reimplemented from [BaseGfxApp](#).

7.16.3.2 `void Simulation::clear ( int )`

7.16.3.3 `void Simulation::display ( )` [virtual]

Call-back function to render all the objects. This function is called repeatedly from glut, and iterates through the objects and calls their respective `display` functions.

Reimplemented from [BaseGfxApp](#).

7.16.3.4 `void Simulation::gluiControl ( int controlID )` [virtual]

Call-back function for user input.

Reimplemented from [BaseGfxApp](#).

7.16.3.5 `void Simulation::initObjects ( )`

Adds objects according to configuration.

7.16.3.6 `void Simulation::keyboard ( unsigned char key, int x, int y )` [virtual]

Call-back function keyboard press.

Reimplemented from [BaseGfxApp](#).

7.16.3.7 `void Simulation::keyboardSpecial ( int key, int x, int y )` [virtual]

Call-back function for special keyboard press.

Reimplemented from [BaseGfxApp](#).

7.16.3.8 `void Simulation::leftMouseDown ( int x, int y )` [virtual]

Call-back function for mouse click.

Reimplemented from [BaseGfxApp](#).

7.16.3.9 `void Simulation::leftMouseUp ( int x, int y )` [virtual]

Call-back function for mouse click.

Reimplemented from [BaseGfxApp](#).

7.16.3.10 void Simulation::middleMouseDown ( int x, int y ) [virtual]

Call-back function for mouse click.

Reimplemented from [BaseGfxApp](#).

7.16.3.11 void Simulation::mouseDragged ( int x, int y ) [virtual]

Call-back function for mouse drag.

Reimplemented from [BaseGfxApp](#).

7.16.3.12 void Simulation::pause ( int )

7.16.3.13 void Simulation::random ( int )

7.16.3.14 void Simulation::reset ( int )

7.16.3.15 void Simulation::resume ( int )

7.16.3.16 void Simulation::s\_addMovingLightSource ( int a ) [static]

7.16.3.17 void Simulation::s\_addObstacle ( int a ) [static]

7.16.3.18 void Simulation::s\_addRobot ( int a ) [static]

7.16.3.19 void Simulation::s\_addRobotTarget ( int a ) [static]

7.16.3.20 void Simulation::s\_addStationaryLightSource ( int a ) [static]

7.16.3.21 void Simulation::s\_clear ( int a ) [static]

7.16.3.22 void Simulation::s\_neuralNetworkFileChanged ( int a ) [static]

7.16.3.23 void Simulation::s\_open ( int a ) [static]

7.16.3.24 void Simulation::s\_pause ( int a ) [static]

7.16.3.25 void Simulation::s\_random ( int a ) [static]

7.16.3.26 void Simulation::s\_refreshConfiguration ( int a ) [static]

7.16.3.27 void Simulation::s\_removeAllLightSource ( int a ) [static]

7.16.3.28 void Simulation::s\_removeAllObstacle ( int a ) [static]

7.16.3.29 void Simulation::s\_removeAllRobotTarget ( int a ) [static]

7.16.3.30 void Simulation::s\_removeLightSource ( int a ) [static]

7.16.3.31 void Simulation::s\_removeObstacle ( int a ) [static]

7.16.3.32 void Simulation::s\_removeRobotTarget ( int a ) [static]

7.16.3.33 void Simulation::s\_reset ( int a ) [static]

7.16.3.34 void Simulation::s\_resume ( int a ) [static]

7.16.3.35 void Simulation::s\_save ( int a ) [static]

7.16.3.36 void Simulation::s\_start ( int a ) [static]

7.16.3.37 void Simulation::start ( int )

7.16.3.38 void Simulation::tryAddMovingLightSource ( )

Adds a moving light to the simulation, or raises an error on failure.

7.16.3.39 void Simulation::tryAddObstacle ( )

Adds a obstacle to the simulation, or raises an error on failure.

7.16.3.40 void Simulation::tryAddRobot ( )

Adds a robot to the simulation, or raises an error on failure.

7.16.3.41 void Simulation::tryAddRobotTarget ( )

Adds a robot and target to the simulation, or raises an error on failure.

7.16.3.42 void Simulation::tryAddStationaryLightSource ( )

Adds a stationary light to the simulation, or raises an error on failure.

7.16.3.43 void Simulation::tryOpen ( )

Loads a simulation, or raises an error on failure.

7.16.3.44 void Simulation::tryRemoveLightSource ( )

Removes a light from the simulation, or raises an error on failure.

7.16.3.45 void Simulation::tryRemoveObstacle ( )

Removes a obstacle from the simulation, or raises an error on failure.

7.16.3.46 void Simulation::tryRemoveRobotTarget ( )

Removes a robot and target from the simulation, or raises an error on failure.

7.16.3.47 void Simulation::trySave ( )

Saves a simulation, or raises an error on failure.

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

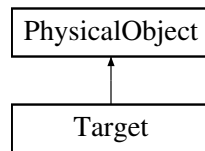
- [Simulation.h](#)
- [Simulation.cpp](#)

## 7.17 Target Class Reference

[Target](#) for the robot to seek.

```
#include <Target.h>
```

Inheritance diagram for Target:



### Public Member Functions

- [Target](#) ()
- [Target](#) (int radius, [Color](#) color, int targetNum=[TARGET](#))
- [Target](#) (int maxRadius, int minRadius, [Color](#) color, int targetNum=[TARGET](#))
- [Target](#) (int radius, [Location](#) loc, [Color](#) color, int targetNum=[TARGET](#))
- bool [handleCollision](#) (int otherId, bool wasHit)
- [~Target](#) ()

### Additional Inherited Members

#### 7.17.1 Detailed Description

[Target](#) for the robot to seek.

#### 7.17.2 Constructor & Destructor Documentation

7.17.2.1 `Target::Target ( )`

7.17.2.2 `Target::Target ( int radius, Color color, int targetNum = TARGET )`

7.17.2.3 `Target::Target ( int maxRadius, int minRadius, Color color, int targetNum = TARGET )`

7.17.2.4 `Target::Target ( int radius, Location loc, Color color, int targetNum = TARGET )`

7.17.2.5 `Target::~~Target ( )`

This is the class destructor.

#### 7.17.3 Member Function Documentation

7.17.3.1 `bool Target::handleCollision ( int otherId, bool wasHit ) [virtual]`

Author

Lucas Kramer This function gives [Target](#) an appropriate reaction when a collision occurs.

**Parameters**

<i>otherId</i>	id of the object that it collides with
<i>wasHit</i>	tells if the object was hit previously

Implements [PhysicalObject](#).

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

- [Target.h](#)
- [Target.cpp](#)



## Chapter 8

# File Documentation

### 8.1 artist.cpp File Reference

```
#include "Color.h"
#include "artist.h"
#include "configuration.h"
#include <GL/glut.h>
#include <GL/glu.h>
#include <math.h>
```

#### Namespaces

- [artist](#)  
*graphical commands*

#### Macros

- [#define \\_USE\\_MATH\\_DEFINES](#)

#### Functions

- void [artist::drawObject](#) ([Location](#) loc, int radius, [Color](#) color)
- void [artist::debugArrow](#) ([Location](#) loc, int orientation)
- void [artist::drawLight](#) ([Location](#) loc, int radius, [Color](#) color)
- void [artist::drawObstacle](#) ([Location](#) loc, int radius)
- void [artist::drawSensor](#) ([Location](#) loc, int orientation, int angle, float intensity)
- void [artist::drawRobot](#) ([Location](#) loc, int radius, int orientation, [Color](#) color, [Color](#) lineColor)

#### 8.1.1 Macro Definition Documentation

##### 8.1.1.1 [#define \\_USE\\_MATH\\_DEFINES](#)

### 8.2 artist.h File Reference

A namespace for OpenGL graphical functions.

```
#include "Color.h"
#include "Location.h"
```

## Namespaces

- [artist](#)  
*graphical commands*

## Functions

- void [artist::debugArrow](#) ([Location](#) loc, int orientation)
- void [artist::drawObject](#) ([Location](#) loc, int radius, [Color](#) color)
- void [artist::drawLight](#) ([Location](#) loc, int radius, [Color](#) color)
- void [artist::drawObstacle](#) ([Location](#) loc, int radius)
- void [artist::drawSensor](#) ([Location](#) loc, int orientation, int angle, float intensity)
- void [artist::drawRobot](#) ([Location](#) loc, int radius, int orientation, [Color](#) color, [Color](#) lineColor)

### 8.2.1 Detailed Description

A namespace for OpenGL graphical functions.

#### Author

Carl Bahn Lucas Kramer

## 8.3 BaseGfxApp.cpp File Reference

Implementation of [BaseGfxApp](#).

```
#include "BaseGfxApp.h"
```

### 8.3.1 Detailed Description

Implementation of [BaseGfxApp](#).

#### Author

CSci3081 Guru Carl Bahn, Lucas Kramer

## 8.4 BaseGfxApp.h File Reference

The basic application class for CSci-3081 project. Uses GLUT and GLUI and wraps them in a nice C++ interface.

```
#include <string>
#include <iostream>
#include <assert.h>
#include <GL/glui.h>
```

## Classes

- class [BaseGfxApp](#)  
*Graphics driver class.*

### 8.4.1 Detailed Description

The basic application class for CSci-3081 project. Uses GLUT and GLUI and wraps them in a nice C++ interface.

#### Author

CSci3081 Guru Carl Bahn, Lucas Kramer

## 8.5 Color.cpp File Reference

Utility functions for handling color.

```
#include "Color.h"
#include "configuration.h"
#include <iostream>
#include <cstdlib>
#include <math.h>
```

### 8.5.1 Detailed Description

Utility functions for handling color.

#### Author

Carl Bahn Lucas Kramer

## 8.6 Color.h File Reference

Definintions for representing any hexadecimal color.

```
#include <iostream>
```

## Classes

- struct [Color](#)  
*This struct holds the representation of a color.*

## Macros

- #define [GET\\_COLOR](#)(COLOR\_VAR)

## Typedefs

- typedef struct [Color](#) [Color](#)  
*This struct holds the representation of a color.*

### 8.6.1 Detailed Description

Definitions for representing any hexadecimal color.

#### Author

Lucas Kramer

### 8.6.2 Macro Definition Documentation

#### 8.6.2.1 #define GET\_COLOR( COLOR\_VAR )

#### Value:

```
(GET_BOOL("USE_HEX_COLORS") ?
    Color(GET_INT(COLOR_VAR "_HEX")) :
    Color(GET_CHAR(COLOR_VAR)))
```

### 8.6.3 Typedef Documentation

#### 8.6.3.1 typedef struct Color Color

This struct holds the representation of a color.

Each field is a float value that should be between 0 and 1. This allows a direct translation from hex colors, with each field being the hex RGB value / 255.

## 8.7 ComplexRobot.cpp File Reference

The representation of a robot within the simulation.

```
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdexcept>
#include "environment.h"
#include "Robot.h"
#include "ComplexRobot.h"
#include "configuration.h"
```

### 8.7.1 Detailed Description

The representation of a robot within the simulation.

#### Author

Lucas Kramer

## 8.8 ComplexRobot.h File Reference

The representation of a robot within the simulation.

```
#include "PhysicalObject.h"
#include "Robot.h"
#include "Sensor.h"
#include "configuration.h"
```

## Classes

- class [ComplexRobot](#)  
*A complex robot with configurable feedback from all sensors.*

### 8.8.1 Detailed Description

The representation of a robot within the simulation.

#### Author

Lucas Kramer

## 8.9 environment.cpp File Reference

```
#include "PhysicalObject.h"
#include "environment.h"
#include "configuration.h"
#include <iostream>
#include <math.h>
```

## Functions

- bool [isTouching](#) ([Location](#) l1, int r1, [Location](#) l2, int r2)

### 8.9.1 Function Documentation

#### 8.9.1.1 bool isTouching ( Location l1, int r1, Location l2, int r2 )

Checks if two objects specified by their Locations and radii

#### Returns

a boolean value that is true when objects are overlapping

## 8.10 environment.h File Reference

```
#include "Location.h"
#include "configuration.h"
#include <unordered_map>
#include <vector>
#include <mutex>
#include <stdexcept>
```

## Classes

- class [ObjectIterator](#)  
*[ObjectIterator](#) class, a simple iterator for the object in environment.*
- class [NoOpenLocationException](#)  
*An exception to be thrown when an open [Location](#) cannot be found.*

## Namespaces

- [environment](#)  
*environment namespace, handles all the objects as a group. Also manages object ids and collision detection*

## Typedefs

- typedef [ObjectIterator](#) [environment::objectIterator](#)

## Functions

- int [environment::addObject](#) ([PhysicalObject](#) \*object)  
*Adds an object to the environment.*
- void [environment::removeObject](#) (int id)  
*Removes an object from the environment.*
- void [environment::clear](#) ()  
*Removes all objects from the environment and resets the id counter.*
- unsigned [environment::getNumObjects](#) ()  
*Gets the number of objects in environment.*
- [PhysicalObject](#) \* [environment::getObject](#) (int id)  
*Finds an object from the environment.*
- [objectIterator](#) [environment::getObjectsBegin](#) ()  
*Gets an iterator to the beginning of the objects.*
- [objectIterator](#) [environment::getObjectsEnd](#) ()  
*Gets an iterator to the end of the objects.*
- bool [environment::isTouchingWall](#) ([Location](#) l, int r)
- bool [environment::isTouchingWall](#) (int id)
- bool [environment::isTouchingObject](#) ([Location](#) l, int r, int id)
- bool [environment::isTouchingHitableObject](#) ([Location](#) l, int r, int id)
- bool [environment::isTouchingObject](#) ([Location](#) l, int r)
- bool [environment::isTouchingHitableObject](#) ([Location](#) l, int r)
- bool [environment::isTouchingObject](#) (int id)
- bool [environment::isTouchingHitableObject](#) (int id)
- bool [environment::isColliding](#) ([Location](#) l, int r)
- bool [environment::isColliding](#) (int id)
- bool [environment::isCollidingWithHitable](#) ([Location](#) l, int r)
- bool [environment::isCollidingWithHitable](#) (int id)
- int [environment::getCollisionId](#) ([Location](#) l, int r, int id)
- int [environment::getHitableCollisionId](#) ([Location](#) l, int r, int id)
- int [environment::getCollisionId](#) ([Location](#) l, int r)
- int [environment::getHitableCollisionId](#) ([Location](#) l, int r)
- int [environment::getCollisionId](#) (int id)
- int [environment::getHitableCollisionId](#) (int id)

## 8.11 LightSource.cpp File Reference

The representation of a target in the simulation.

```
#include <stdlib.h>
#include <math.h>
#include "artist.h"
#include "environment.h"
#include "LightSource.h"
#include "configuration.h"
```

### 8.11.1 Detailed Description

The representation of a target in the simulation.

Author

Himawan Go Lucas Kramer

## 8.12 LightSource.h File Reference

The representation of an obstacle in the simulation.

```
#include "PhysicalObject.h"
#include "configuration.h"
```

### Classes

- class [LightSource](#)  
*LightSource for the robot to seek.*

### 8.12.1 Detailed Description

The representation of an obstacle in the simulation.

Author

Himawan Go Lucas Kramer

## 8.13 Location.h File Reference

### Classes

- struct [Location](#)  
*an xy position on the screen, approximately in pixels x is horizontal from the left and y is vertical from the bottom*

### Typedefs

- typedef struct [Location](#) [Location](#)  
*an xy position on the screen, approximately in pixels x is horizontal from the left and y is vertical from the bottom*

### 8.13.1 Typedef Documentation

#### 8.13.1.1 typedef struct Location Location

an xy position on the screen, approximately in pixels x is horizontal from the left and y is vertical from the bottom

## 8.14 main.cpp File Reference

The main function to execute the simulation program.

```
#include <stdlib.h>
#include <time.h>
#include "OptimizeSimulation.h"
#include "Simulation.h"
#include "configuration.h"
```

### Macros

- #define `DEFAULT_CONFIG` `"../config/default"`

### Functions

- int `main` (int `argc`, char \*`argv`[])

#### 8.14.1 Detailed Description

The main function to execute the simulation program.

#### Author

Lucas Kramer  
Carl Bahn  
Himawan Go

#### 8.14.2 Macro Definition Documentation

##### 8.14.2.1 #define `DEFAULT_CONFIG` `"../config/default"`

#### 8.14.3 Function Documentation

##### 8.14.3.1 int `main` ( int `argc`, char \* `argv`[] )

Main function to execute the simulation

## 8.15 mainpage.h File Reference

## 8.16 NeuralNetwork.cpp File Reference

A feed-forward neural network class.



```
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <mutex>
#include <fstream>
#include <sstream>
#include <string>
#include <string.h>
#include <boost/regex.hpp>
#include <boost/algorithm/string.hpp>
#include <boost/algorithm/string/regex.hpp>
#include "NeuralNetwork.h"
```

### 8.16.1 Detailed Description

A feed-forward neural network class.

#### Author

Lucas Kramer

## 8.17 NeuralNetwork.h File Reference

A feed-forward neural network class.

```
#include <vector>
#include <mutex>
```

### Classes

- class [NeuralNetwork](#)  
*A representation of a neural network.*

### 8.17.1 Detailed Description

A feed-forward neural network class.

#### Author

Lucas Kramer

## 8.18 NeuralNetworkRobot.cpp File Reference

The representation of a neural network robot within the simulation.

```
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdexcept>
#include "environment.h"
#include "Robot.h"
#include "NeuralNetworkRobot.h"
#include "NeuralNetwork.h"
#include "configuration.h"
```

### 8.18.1 Detailed Description

The representation of a neural network robot within the simulation.

Author

Lucas Kramer

## 8.19 NeuralNetworkRobot.h File Reference

The representation of a neural network robot.

```
#include "PhysicalObject.h"
#include "Robot.h"
#include "Sensor.h"
#include "NeuralNetwork.h"
#include "configuration.h"
```

### Classes

- class [NeuralNetworkRobot](#)

*A robot controlled by a neural network with inputs from the sensors.*

### 8.19.1 Detailed Description

The representation of a neural network robot.

Author

Lucas Kramer

## 8.20 Obstacle.cpp File Reference

The representation of an obstacle in the simulation.

```
#include <stdlib.h>
#include <math.h>
#include "environment.h"
#include "Obstacle.h"
#include "configuration.h"
#include "artist.h"
```

### 8.20.1 Detailed Description

The representation of an obstacle in the simulation.

Author

Himawan Go Lucas Kramer

## 8.21 Obstacle.h File Reference

The representation of an obstacle in the simulation.

```
#include "PhysicalObject.h"
#include "configuration.h"
```

### Classes

- class [Obstacle](#)  
*Obstacle for the robot to hit/avoid.*

### 8.21.1 Detailed Description

The representation of an obstacle in the simulation.

#### Author

Himawan Go Lucas Kramer

## 8.22 OptimizeSimulation.cpp File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <algorithm>
#include <vector>
#include <mutex>
#include <stdexcept>
#include <sstream>
#include <fstream>
#include <iostream>
#include <climits>
#include <csignal>
#include <sys/stat.h>
#include <boost/interprocess/sync/named_upgradable_mutex.hpp>
#include "Robot.h"
#include "SimpleRobot.h"
#include "ComplexRobot.h"
#include "NeuralNetworkRobot.h"
#include "NeuralNetwork.h"
#include "Target.h"
#include "Obstacle.h"
#include "LightSource.h"
#include "Location.h"
#include "configuration.h"
#include "environment.h"
#include "util.h"
#include "OptimizeSimulation.h"
```

## 8.23 OptimizeSimulation.h File Reference

[Robot](#) simultaion for optimizing the [NeuralNetworkRobot](#).

```
#include <vector>
#include <boost/interprocess/sync/named_upgradable_mutex.hpp>
#include "Color.h"
#include "Location.h"
#include "configuration.h"
#include "environment.h"
#include "util.h"
#include "PhysicalObject.h"
#include "NeuralNetwork.h"
```

### Classes

- class [OptimizeSimulation](#)  
*[OptimizeSimulation](#) class, sets up environments and robots.*

#### 8.23.1 Detailed Description

[Robot](#) simultaion for optimizing the [NeuralNetworkRobot](#).

#### Author

Lucas Kramer

## 8.24 PhysicalObject.cpp File Reference

The representation of any object within the simulation.

```
#include <stdlib.h>
#include <iostream>
#include <math.h>
#include <assert.h>
#include <unistd.h>
#include <stdexcept>
#include "artist.h"
#include "PhysicalObject.h"
#include "environment.h"
#include "configuration.h"
```

### Macros

- `#define \_USE\_MATH\_DEFINES`

#### 8.24.1 Detailed Description

The representation of any object within the simulation.

#### Author

Lucas Kramer Himawan Go, Carl Bahn

## 8.24.2 Macro Definition Documentation

### 8.24.2.1 #define \_USE\_MATH\_DEFINES

## 8.25 PhysicalObject.h File Reference

The representation of any object within the simulation.

```
#include "Color.h"
#include "configuration.h"
#include "Location.h"
```

### Classes

- class [PhysicalObject](#)  
*This is a common superclass for all objects.*

### Enumerations

- enum [ObjectType](#) {  
    [ROBOT](#), [TARGET](#), [OBSTACLE](#), [LIGHT](#),  
    [LAST](#) = [LIGHT](#) }

### 8.25.1 Detailed Description

The representation of any object within the simulation.

#### Author

Lucas Kramer Himawan Go, Carl Bahn

## 8.25.2 Enumeration Type Documentation

### 8.25.2.1 enum ObjectType

#### Enumerator

***ROBOT***  
***TARGET***  
***OBSTACLE***  
***LIGHT***  
***LAST***

## 8.26 Robot.cpp File Reference

The representation of robot within the simulation.

```
#include <stdexcept>
#include <math.h>
#include "artist.h"
#include "environment.h"
#include "Robot.h"
#include "configuration.h"
```

### 8.26.1 Detailed Description

The representation of robot within the simulation.

#### Author

Lucas Kramer Himawan Go, Carl Bahn

## 8.27 Robot.h File Reference

The representation of robot within the simulation.

```
#include "PhysicalObject.h"
#include "Sensor.h"
#include "configuration.h"
```

### Classes

- class [Robot](#)  
*[Robot](#) that moves around the window and bumps into obstacles.*

### Enumerations

- enum [RobotType](#) { [SIMPLE](#), [COMPLEX](#), [NEURAL\\_NETWORK](#) }

### 8.27.1 Detailed Description

The representation of robot within the simulation.

#### Author

Himawan Go Lucas Kramer

### 8.27.2 Enumeration Type Documentation

#### 8.27.2.1 enum RobotType

##### Enumerator

***SIMPLE***

***COMPLEX***

***NEURAL\_NETWORK***

## 8.28 Sensor.cpp File Reference

A sensor that detects certain objects in the environment.

```
#include "artist.h"
#include "Sensor.h"
#include "environment.h"
#include "configuration.h"
#include "PhysicalObject.h"
#include <iostream>
#include <math.h>
#include <stdexcept>
```

## Macros

- `#define \_USE\_MATH\_DEFINES`

### 8.28.1 Detailed Description

A sensor that detects certain objects in the environment.

#### Author

Himawan Go Carl Bahn Lucas Kramer

### 8.28.2 Macro Definition Documentation

#### 8.28.2.1 `#define \_USE\_MATH\_DEFINES`

## 8.29 Sensor.h File Reference

A sensor that detects certain objects in the environment.

```
#include "Color.h"
#include "Location.h"
#include "configuration.h"
#include "PhysicalObject.h"
```

## Classes

- class [Sensor](#)

### 8.29.1 Detailed Description

A sensor that detects certain objects in the environment.

#### Author

Himawan Go  
Carl Bahn

## 8.30 SimpleRobot.cpp File Reference

The representation of robot within the simulation.

```
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdexcept>
#include "environment.h"
#include "Robot.h"
#include "SimpleRobot.h"
#include "configuration.h"
```

### 8.30.1 Detailed Description

The representation of robot within the simulation.

#### Author

Lucas Kramer

## 8.31 SimpleRobot.h File Reference

The representation of a simple robot.

```
#include "PhysicalObject.h"
#include "Robot.h"
#include "Sensor.h"
#include "configuration.h"
```

### Classes

- class [SimpleRobot](#)

*A simple robot with uncrossed feedback.*

### 8.31.1 Detailed Description

The representation of a simple robot.



**Author**

Lucas Kramer

## 8.32 Simulation.cpp File Reference

Implementation for the main application class of the robot simulation.

```
#include <unistd.h>
#include <limits.h>
#include <deque>
#include <mutex>
#include <stdexcept>
#include <sstream>
#include <iostream>
#include <fstream>
#include <dirent.h>
#include <sys/stat.h>
#include "PhysicalObject.h"
#include "Robot.h"
#include "SimpleRobot.h"
#include "ComplexRobot.h"
#include "NeuralNetworkRobot.h"
#include "Target.h"
#include "Obstacle.h"
#include "LightSource.h"
#include "Location.h"
#include "configuration.h"
#include "environment.h"
#include "util.h"
#include "Simulation.h"
```

**Functions**

- string [ftos](#) (float f)

### 8.32.1 Detailed Description

Implementation for the main application class of the robot simulation.

**Author**

Lucas Kramer  
Lucas Kramer Xi Zhang, Carl Bahn

### 8.32.2 Function Documentation

#### 8.32.2.1 string ftos ( float f )

## 8.33 Simulation.h File Reference

Main application class for the robot simulation.

```
#include "Color.h"
#include "Location.h"
#include "configuration.h"
#include "environment.h"
#include "util.h"
#include "BaseGfxApp.h"
#include "PhysicalObject.h"
```

## Classes

- class [Simulation](#)  
*Simulation class, sets up the GUI and the drawing environment.*

### 8.33.1 Detailed Description

Main application class for the robot simulation.

#### Author

Lucas Kramer Xi Zhang, Carl Bahn

## 8.34 Target.cpp File Reference

The representation of a target in the simulation.

```
#include <stdlib.h>
#include <math.h>
#include "environment.h"
#include "Target.h"
#include "configuration.h"
```

### 8.34.1 Detailed Description

The representation of a target in the simulation.

#### Author

Himawan Go Lucas Kramer

## 8.35 Target.h File Reference

The representation of an obstacle in the simulation.

```
#include "PhysicalObject.h"
```

## Classes

- class [Target](#)  
*Target for the robot to seek.*

### 8.35.1 Detailed Description

The representation of an obstacle in the simulation.

#### Author

Himawan Go Lucas Kramer

## 8.36 util.cpp File Reference

Implmentation of functions for the simulation.

```
#include <unistd.h>
#include <stack>
#include <queue>
#include <mutex>
#include <stdexcept>
#include <sstream>
#include <iostream>
#include <fstream>
#include <GL/glui.h>
#include <assert.h>
#include "Robot.h"
#include "SimpleRobot.h"
#include "ComplexRobot.h"
#include "NeuralNetwork.h"
#include "NeuralNetworkRobot.h"
#include "Target.h"
#include "Obstacle.h"
#include "LightSource.h"
#include "Location.h"
#include "configuration.h"
#include "environment.h"
#include "util.h"
```

#### Variables

- std::mutex [objectsMutex](#)
- stack< int > [robots](#)
- stack< int > [targets](#)
- stack< int > [lights](#)
- stack< int > [obstacles](#)
- int [colorNum](#) = 0

### 8.36.1 Detailed Description

Implmentation of functions for the simulation.

#### Author

Lucas Kramer

## 8.36.2 Variable Documentation

8.36.2.1 `int colorNum = 0`

8.36.2.2 `stack<int> lights`

8.36.2.3 `std::mutex objectsMutex`

8.36.2.4 `stack<int> obstacles`

8.36.2.5 `stack<int> robots`

8.36.2.6 `stack<int> targets`

## 8.37 util.h File Reference

Helper functions for the simulation.

```
#include "NeuralNetwork.h"
```

### Namespaces

- [util](#)

*util namespace, contains helper functions to add and remove robots*

### Functions

- void [util::reset](#) ()

*Removes all objects and resets to the initial state.*

- void [util::display](#) ()

*Function to render all the objects. This function is called repeatedly from the simulation, and iterates through the objects and calls their respective display functions.*

- void [util::advance](#) ()

*Function to update the positions of all objects. This function is called repeatedly from the simulation, and iterates through the objects and calls their respective updatePosition functions.*

- [Color util::newColor](#) ()

*Gets a new, unused color.*

- int [util::getNumRobotsTargets](#) ()

*Gets the number of robots/target pairs.*

- int [util::getNumLights](#) ()

*Gets the number of lights.*

- int [util::getNumObstacles](#) ()

*Gets the number of obstacles.*

- bool [util::addRobotTarget](#) (int robotType, int lightSensorConnectionPattern, int robotSensorConnectionPattern, int obstacleSensorConnectionPattern, int targetSensorConnectionPattern, float lightSensorScale, float robotSensorScale, float obstacleSensorScale, float targetSensorScale, int initialSpeed, std::string neuralNetworkFile)

*Adds a robot and paired target to the simulation.*

- bool [util::addRobot](#) (int robotType, int lightSensorConnectionPattern, int robotSensorConnectionPattern, int obstacleSensorConnectionPattern, int targetSensorConnectionPattern, float lightSensorScale, float robotSensorScale, float obstacleSensorScale, float targetSensorScale, int initialSpeed, std::string neuralNetworkFile)

- Adds a robot to the simulation.*
  - bool `util::addNeuralNetworkRobotTarget` (const `NeuralNetwork` &network)  
*Adds a neural network robot and a paired target to the simulation.*
- bool `util::addStationaryLightSource` ()  
*Adds a light to the simulation.*
- bool `util::addMovingLightSource` ()  
*Adds a light to the simulation.*
- bool `util::addObstacle` ()  
*Adds a obstacle to the simulation.*
- bool `util::copy` (int id, `Location` loc)  
*Copies an object to a new `Location`.*
- bool `util::removeRobotTarget` ()  
*Removes a robot from the simulation.*
- bool `util::removeLightSource` ()  
*Removes a light from the simulation.*
- bool `util::removeObstacle` ()  
*Removes a obstacle from the simulation.*
- void `util::removeAllRobotTarget` ()  
*Removes all robots.*
- void `util::removeAllLightSource` ()  
*Removes all lights.*
- void `util::removeAllObstacle` ()  
*Removes all obstacles.*
- bool `util::open` (std::string filename)  
*Loads a simulation from a file.*
- bool `util::save` (std::string filename)  
*Saves the current state to a file.*

### 8.37.1 Detailed Description

Helper functions for the simulation.

Author

Lucas Kramer

# Index

- ~BaseGfxApp
  - BaseGfxApp, [30](#)
- ~ComplexRobot
  - ComplexRobot, [36](#)
- ~LightSource
  - LightSource, [39](#)
- ~NeuralNetworkRobot
  - NeuralNetworkRobot, [44](#)
- ~ObjectIterator
  - ObjectIterator, [46](#)
- ~Obstacle
  - Obstacle, [48](#)
- ~OptimizeSimulation
  - OptimizeSimulation, [49](#)
- ~PhysicalObject
  - PhysicalObject, [51](#)
- ~Robot
  - Robot, [59](#)
- ~SimpleRobot
  - SimpleRobot, [65](#)
- ~Simulation
  - Simulation, [67](#)
- ~Target
  - Target, [71](#)
- \_USE\_MATH\_DEFINES
  - artist.cpp, [73](#)
  - PhysicalObject.cpp, [85](#)
  - Sensor.cpp, [87](#)
- addMovingLightSource
  - util, [23](#)
- addNeuralNetworkRobotTarget
  - util, [23](#)
- addObject
  - environment, [15](#)
- addObstacle
  - util, [24](#)
- addRobot
  - util, [24](#)
- addRobotTarget
  - util, [24](#)
- addStationaryLightSource
  - util, [25](#)
- advance
  - BaseGfxApp, [30](#)
  - Simulation, [68](#)
  - util, [25](#)
- artist, [13](#)
  - debugArrow, [13](#)
  - drawLight, [13](#)
  - drawObject, [13](#)
  - drawObstacle, [14](#)
  - drawRobot, [14](#)
  - drawSensor, [14](#)
- artist.cpp, [73](#)
  - \_USE\_MATH\_DEFINES, [73](#)
- artist.h, [73](#)
- BaseGfxApp, [29](#)
  - ~BaseGfxApp, [30](#)
  - advance, [30](#)
  - BaseGfxApp, [30](#)
  - BaseGfxApp, [30](#)
  - display, [30](#)
  - glui, [30](#)
  - gluiControl, [31](#)
  - graphicsBegin, [31](#)
  - graphicsTimer, [31](#)
  - handle, [31](#)
  - height, [31](#)
  - keyboard, [31](#)
  - keyboardSpecial, [31](#)
  - keyboardSpecialUp, [31](#)
  - keyboardUp, [31](#)
  - leftMouseDown, [31](#)
  - leftMouseUp, [31](#)
  - m\_drag, [32](#)
  - m\_glui, [32](#)
  - m\_glutWindowHandle, [32](#)
  - m\_height, [32](#)
  - m\_width, [32](#)
  - middleMouseDown, [31](#)
  - middleMouseUp, [31](#)
  - mouseDragged, [31](#)
  - mouseMoved, [31](#)
  - reshape, [32](#)
  - rightMouseDown, [32](#)
  - rightMouseUp, [32](#)
  - runMainLoop, [32](#)
  - s\_currentApp, [32](#)
  - s\_draw, [32](#)
  - s\_gluicallback, [32](#)
  - s\_glutInitialized, [32](#)
  - s\_keyboard, [32](#)
  - s\_keyboardspecial, [32](#)
  - s\_keyboardspecialup, [32](#)
  - s\_keyboardup, [32](#)
  - s\_mousebtn, [32](#)
  - s\_mousemotion, [32](#)
  - s\_reshape, [32](#)

- setCaption, 32
- width, 32
- BaseGfxApp.cpp, 74
- BaseGfxApp.h, 74
- blue
  - Color, 35
- COMPLEX
  - Robot.h, 86
- clear
  - environment, 16
  - Simulation, 68
- Color, 33
  - blue, 35
  - Color, 33, 34
  - Color.h, 76
  - green, 35
  - isSimilar, 34
  - operator<<, 35
  - operator==, 35
  - red, 35
- Color.cpp, 75
- Color.h, 75
  - Color, 76
  - GET\_COLOR, 76
- colorNum
  - util.cpp, 92
- combine
  - NeuralNetwork, 41
- ComplexRobot, 35
  - ~ComplexRobot, 36
  - ComplexRobot, 36
  - ComplexRobot, 36
  - defaultSpeed, 37
  - enableLightSensors, 37
  - enableObstacleSensors, 37
  - enableRobotSensors, 37
  - enableTargetSensors, 37
  - getLeftSpeed, 37
  - getRightSpeed, 37
  - lightSensorScale, 37
  - lightSensorsCrossed, 37
  - obstacleSensorScale, 37
  - obstacleSensorsCrossed, 37
  - robotSensorScale, 37
  - robotSensorsCrossed, 37
  - targetSensorScale, 37
  - targetSensorsCrossed, 37
- ComplexRobot.cpp, 76
- ComplexRobot.h, 76
- compute
  - NeuralNetwork, 41
- copy
  - util, 25
- DEFAULT\_CONFIG
  - main.cpp, 80
- debugArrow
  - artist, 13
- defaultSpeed
  - ComplexRobot, 37
- display
  - BaseGfxApp, 30
  - LightSource, 39
  - PhysicalObject, 51
  - Robot, 59
  - Sensor, 62
  - Simulation, 68
  - util, 25
- drawLight
  - artist, 13
- drawObject
  - artist, 13
- drawObstacle
  - artist, 14
- drawRobot
  - artist, 14
- drawSensor
  - artist, 14
- enableLightSensors
  - ComplexRobot, 37
- enableObstacleSensors
  - ComplexRobot, 37
- enableRobotSensors
  - ComplexRobot, 37
- enableTargetSensors
  - ComplexRobot, 37
- environment, 14
  - addObject, 15
  - clear, 16
  - getCollisionId, 16
  - getHitableCollisionId, 17
  - getNumObjects, 18
  - getObject, 18
  - getObjectsBegin, 18
  - getObjectsEnd, 18
  - isColliding, 18, 19
  - isCollidingWithHitable, 19
  - isTouchingHitableObject, 20
  - isTouchingObject, 20, 21
  - isTouchingWall, 21
  - objectIterator, 15
  - removeObject, 22
- environment.cpp, 77
  - isTouching, 77
- environment.h, 77
- environment::getObjectsBegin
  - ObjectIterator, 47
- environment::getObjectsEnd
  - ObjectIterator, 47
- filename
  - NeuralNetworkRobot, 45
- findOpenLocation
  - PhysicalObject, 51
- forceSetPosition
  - PhysicalObject, 53

- forceTranslate
  - PhysicalObject, 53
- ftos
  - Simulation.cpp, 89
- GET\_COLOR
  - Color.h, 76
- getCollisionId
  - environment, 16
- getColor
  - PhysicalObject, 53
- getHitableCollisionId
  - environment, 17
- getId
  - PhysicalObject, 53
- getLeftSpeed
  - ComplexRobot, 37
  - NeuralNetworkRobot, 44
  - Robot, 59
  - SimpleRobot, 65
- getLineColor
  - Robot, 59
- getLocation
  - PhysicalObject, 53
- getNumLights
  - util, 26
- getNumObjects
  - environment, 18
- getNumObstacles
  - util, 26
- getNumRobotsTargets
  - util, 26
- getObject
  - environment, 18
- getObjectsBegin
  - environment, 18
- getObjectsEnd
  - environment, 18
- getOrientation
  - PhysicalObject, 53
  - Sensor, 62
- getPerformance
  - OptimizeSimulation, 49
- getPerformanceMaze
  - OptimizeSimulation, 50
- getPerformanceObstacles
  - OptimizeSimulation, 50
- getPerformanceRandomRepeated
  - OptimizeSimulation, 50
- getPosition
  - Sensor, 62
- getRadius
  - PhysicalObject, 53
- getRightSpeed
  - ComplexRobot, 37
  - NeuralNetworkRobot, 44
  - Robot, 59
  - SimpleRobot, 65
- getSpeed
  - PhysicalObject, 54
- getTarget
  - Robot, 59
- getTypeDetected
  - Sensor, 62
- getViewAngle
  - Sensor, 62
- getXPosition
  - PhysicalObject, 54
  - Sensor, 62
- getYPosition
  - PhysicalObject, 54
  - Sensor, 63
- glui
  - BaseGfxApp, 30
- gluiControl
  - BaseGfxApp, 31
  - Simulation, 68
- graphicsBegin
  - BaseGfxApp, 31
- graphicsTimer
  - BaseGfxApp, 31
- green
  - Color, 35
- handle
  - BaseGfxApp, 31
- handleCollision
  - LightSource, 39
  - Obstacle, 48
  - PhysicalObject, 54
  - Robot, 60
  - Target, 71
- hasEqualPosition
  - PhysicalObject, 54
- height
  - BaseGfxApp, 31
- initObjects
  - Simulation, 68
- isColliding
  - environment, 18, 19
- isCollidingWithHitable
  - environment, 19
- isHitable
  - PhysicalObject, 57
- isSimilar
  - Color, 34
- isTouching
  - environment.cpp, 77
- isTouchingHitableObject
  - environment, 20
- isTouchingObject
  - environment, 20, 21
- isTouchingWall
  - environment, 21
- keyboard
  - BaseGfxApp, 31



- Simulation, 68
- keyboardSpecial
  - BaseGfxApp, 31
  - Simulation, 68
- keyboardSpecialUp
  - BaseGfxApp, 31
- keyboardUp
  - BaseGfxApp, 31
- LAST
  - PhysicalObject.h, 85
- LIGHT
  - PhysicalObject.h, 85
- leftMouseDown
  - BaseGfxApp, 31
  - Simulation, 68
- leftMouseUp
  - BaseGfxApp, 31
  - Simulation, 68
- lightSensorScale
  - ComplexRobot, 37
- lightSensorsCrossed
  - ComplexRobot, 37
- LightSource, 37
  - ~LightSource, 39
  - display, 39
  - handleCollision, 39
  - LightSource, 38, 39
  - LightSource, 38, 39
- LightSource.cpp, 79
- LightSource.h, 79
- lights
  - util.cpp, 92
- load
  - NeuralNetwork, 43
- Location, 40
  - Location, 40
  - Location.h, 80
  - operator==, 40
  - x, 40
  - y, 40
- Location.h, 79
  - Location, 80
- m\_drag
  - BaseGfxApp, 32
- m\_glui
  - BaseGfxApp, 32
- m\_glutWindowHandle
  - BaseGfxApp, 32
- m\_height
  - BaseGfxApp, 32
- m\_width
  - BaseGfxApp, 32
- main
  - main.cpp, 80
- main.cpp, 80
  - DEFAULT\_CONFIG, 80
  - main, 80
- mainpage.h, 80
- middleMouseDown
  - BaseGfxApp, 31
  - Simulation, 68
- middleMouseUp
  - BaseGfxApp, 31
- mouseDragged
  - BaseGfxApp, 31
  - Simulation, 69
- mouseMoved
  - BaseGfxApp, 31
- mutate
  - NeuralNetwork, 43
- NEURAL\_NETWORK
  - Robot.h, 86
- NeuralNetwork, 41
  - combine, 41
  - compute, 41
  - load, 43
  - mutate, 43
  - NeuralNetwork, 41
  - NeuralNetwork, 41
  - write, 43
- NeuralNetwork.cpp, 80
- NeuralNetwork.h, 81
- NeuralNetworkRobot, 43
  - ~NeuralNetworkRobot, 44
  - filename, 45
  - getLeftSpeed, 44
  - getRightSpeed, 44
  - NeuralNetworkRobot, 44
  - NeuralNetworkRobot, 44
- NeuralNetworkRobot.cpp, 81
- NeuralNetworkRobot.h, 82
- newColor
  - util, 26
- NoOpenLocationException, 45
  - NoOpenLocationException, 45
  - NoOpenLocationException, 45
- OBSTACLE
  - PhysicalObject.h, 85
- ObjectIterator, 45
  - ~ObjectIterator, 46
  - environment::getObjectsBegin, 47
  - environment::getObjectsEnd, 47
  - ObjectIterator, 46
  - ObjectIterator, 46
  - operator<, 47
  - operator>, 47
  - operator\*, 46
  - operator++, 46
  - operator==, 47
- objectIterator
  - environment, 15
- ObjectType
  - PhysicalObject.h, 85
- objectType

- PhysicalObject, 57
- objectsMutex
  - util.cpp, 92
- Obstacle, 47
  - ~Obstacle, 48
  - handleCollision, 48
  - Obstacle, 47, 48
- Obstacle.cpp, 82
- Obstacle.h, 83
- obstacleSensorScale
  - ComplexRobot, 37
- obstacleSensorsCrossed
  - ComplexRobot, 37
- obstacles
  - util.cpp, 92
- open
  - util, 26
- operator<
  - ObjectIterator, 47
- operator<<
  - Color, 35
- operator>
  - ObjectIterator, 47
- operator\*
  - ObjectIterator, 46
- operator++
  - ObjectIterator, 46
- operator==
  - Color, 35
  - Location, 40
  - ObjectIterator, 47
- OptimizeSimulation, 49
  - ~OptimizeSimulation, 49
  - getPerformance, 49
  - getPerformanceMaze, 50
  - getPerformanceObstacles, 50
  - getPerformanceRandomRepeated, 50
  - OptimizeSimulation, 49
  - OptimizeSimulation, 49
  - runMainLoop, 50
  - stop, 50
- OptimizeSimulation.cpp, 83
- OptimizeSimulation.h, 84
- pause
  - Simulation, 69
- PhysicalObject.h
  - LAST, 85
  - LIGHT, 85
  - OBSTACLE, 85
  - ROBOT, 85
  - TARGET, 85
- PhysicalObject, 50
  - ~PhysicalObject, 51
  - display, 51
  - findOpenLocation, 51
  - forceSetPosition, 53
  - forceTranslate, 53
  - getColor, 53
  - getId, 53
  - getLocation, 53
  - getOrientation, 53
  - getRadius, 53
  - getSpeed, 54
  - getXPosition, 54
  - getYPosition, 54
  - handleCollision, 54
  - hasEqualPosition, 54
  - isHitable, 57
  - objectType, 57
  - PhysicalObject, 51
  - PhysicalObject, 51
  - pointTo, 55
  - reorient, 55
  - rotate, 55
  - setColor, 55
  - setLocation, 55
  - setOrientation, 56
  - setPosition, 56
  - setRadius, 56
  - setSpeed, 56
  - translate, 56
  - update, 56
  - updateMembers, 57
  - updatePosition, 57
- PhysicalObject.cpp, 84
- PhysicalObject.h, 85
  - ObjectType, 85
- pointTo
  - PhysicalObject, 55
- ROBOT
  - PhysicalObject.h, 85
- random
  - Simulation, 69
- red
  - Color, 35
- removeAllLightSource
  - util, 26
- removeAllObstacle
  - util, 26
- removeAllRobotTarget
  - util, 26
- removeLightSource
  - util, 26
- removeObject
  - environment, 22
- removeObstacle
  - util, 27
- removeRobotTarget
  - util, 27
- reorient
  - PhysicalObject, 55
- reset
  - Simulation, 69
  - util, 27
- reshape
  - BaseGfxApp, 32

- resume
  - Simulation, 69
- rightMouseDown
  - BaseGfxApp, 32
- rightMouseUp
  - BaseGfxApp, 32
- Robot, 57
  - ~Robot, 59
  - display, 59
  - getLeftSpeed, 59
  - getLineColor, 59
  - getRightSpeed, 59
  - getTarget, 59
  - handleCollision, 60
  - Robot, 58
  - robotType, 61
  - setLineColor, 60
  - setTarget, 60
  - update, 60
  - updateMembers, 60
- Robot.h
  - COMPLEX, 86
  - NEURAL\_NETWORK, 86
  - SIMPLE, 86
- Robot.cpp, 85
- Robot.h, 86
  - RobotType, 86
- robotSensorScale
  - ComplexRobot, 37
- robotSensorsCrossed
  - ComplexRobot, 37
- RobotType
  - Robot.h, 86
- robotType
  - Robot, 61
- robots
  - util.cpp, 92
- rotate
  - PhysicalObject, 55
- runMainLoop
  - BaseGfxApp, 32
  - OptimizeSimulation, 50
- SIMPLE
  - Robot.h, 86
- s\_addMovingLightSource
  - Simulation, 69
- s\_addObstacle
  - Simulation, 69
- s\_addRobot
  - Simulation, 69
- s\_addRobotTarget
  - Simulation, 69
- s\_addStationaryLightSource
  - Simulation, 69
- s\_clear
  - Simulation, 69
- s\_currentApp
  - BaseGfxApp, 32
- s\_draw
  - BaseGfxApp, 32
- s\_gluicallback
  - BaseGfxApp, 32
- s\_glutInitialized
  - BaseGfxApp, 32
- s\_keyboard
  - BaseGfxApp, 32
- s\_keyboardspecial
  - BaseGfxApp, 32
- s\_keyboardspecialup
  - BaseGfxApp, 32
- s\_keyboardup
  - BaseGfxApp, 32
- s\_mousebtn
  - BaseGfxApp, 32
- s\_mousemotion
  - BaseGfxApp, 32
- s\_neuralNetworkFileChanged
  - Simulation, 69
- s\_open
  - Simulation, 69
- s\_pause
  - Simulation, 69
- s\_random
  - Simulation, 69
- s\_refreshConfiguration
  - Simulation, 69
- s\_removeAllLightSource
  - Simulation, 69
- s\_removeAllObstacle
  - Simulation, 69
- s\_removeAllRobotTarget
  - Simulation, 69
- s\_removeLightSource
  - Simulation, 69
- s\_removeObstacle
  - Simulation, 69
- s\_removeRobotTarget
  - Simulation, 69
- s\_reset
  - Simulation, 69
- s\_reshape
  - BaseGfxApp, 32
- s\_resume
  - Simulation, 69
- s\_save
  - Simulation, 70
- s\_start
  - Simulation, 70
- save
  - util, 27
- sense
  - Sensor, 63
- Sensor, 61
  - display, 62
  - getOrientation, 62
  - getPosition, 62

- getTypeDetected, 62
  - getViewAngle, 62
  - getXPosition, 62
  - getYPosition, 63
  - sense, 63
  - Sensor, 61
  - setOrientation, 63
  - setPosition, 63
  - setTypeDetected, 64
  - setViewAngle, 64
  - updatePosition, 64
- Sensor.cpp, 86
- \_USE\_MATH\_DEFINES, 87
- Sensor.h, 87
- setCaption
  - BaseGfxApp, 32
- setColor
  - PhysicalObject, 55
- setLineColor
  - Robot, 60
- setLocation
  - PhysicalObject, 55
- setOrientation
  - PhysicalObject, 56
  - Sensor, 63
- setPosition
  - PhysicalObject, 56
  - Sensor, 63
- setRadius
  - PhysicalObject, 56
- setSpeed
  - PhysicalObject, 56
- setTarget
  - Robot, 60
- setTypeDetected
  - Sensor, 64
- setViewAngle
  - Sensor, 64
- SimpleRobot, 64
  - ~SimpleRobot, 65
  - getLeftSpeed, 65
  - getRightSpeed, 65
  - SimpleRobot, 65
  - SimpleRobot, 65
- SimpleRobot.cpp, 87
- SimpleRobot.h, 88
- Simulation, 65
  - ~Simulation, 67
  - advance, 68
  - clear, 68
  - display, 68
  - gluiControl, 68
  - initObjects, 68
  - keyboard, 68
  - keyboardSpecial, 68
  - leftMouseDown, 68
  - leftMouseUp, 68
  - middleMouseDown, 68
  - mouseDragged, 69
  - pause, 69
  - random, 69
  - reset, 69
  - resume, 69
  - s\_addMovingLightSource, 69
  - s\_addObstacle, 69
  - s\_addRobot, 69
  - s\_addRobotTarget, 69
  - s\_addStationaryLightSource, 69
  - s\_clear, 69
  - s\_neuralNetworkFileChanged, 69
  - s\_open, 69
  - s\_pause, 69
  - s\_random, 69
  - s\_refreshConfiguration, 69
  - s\_removeAllLightSource, 69
  - s\_removeAllObstacle, 69
  - s\_removeAllRobotTarget, 69
  - s\_removeLightSource, 69
  - s\_removeObstacle, 69
  - s\_removeRobotTarget, 69
  - s\_reset, 69
  - s\_resume, 69
  - s\_save, 70
  - s\_start, 70
  - Simulation, 67
  - start, 70
  - tryAddMovingLightSource, 70
  - tryAddObstacle, 70
  - tryAddRobot, 70
  - tryAddRobotTarget, 70
  - tryAddStationaryLightSource, 70
  - tryOpen, 70
  - tryRemoveLightSource, 70
  - tryRemoveObstacle, 70
  - tryRemoveRobotTarget, 70
  - trySave, 70
- Simulation.cpp, 89
  - ftos, 89
- Simulation.h, 89
- start
  - Simulation, 70
- stop
  - OptimizeSimulation, 50
- TARGET
  - PhysicalObject.h, 85
- Target, 71
  - ~Target, 71
  - handleCollision, 71
  - Target, 71
- Target.cpp, 90
- Target.h, 90
- targetSensorScale
  - ComplexRobot, 37
- targetSensorsCrossed
  - ComplexRobot, 37
- targets

- util.cpp, [92](#)
- translate
  - PhysicalObject, [56](#)
- tryAddMovingLightSource
  - Simulation, [70](#)
- tryAddObstacle
  - Simulation, [70](#)
- tryAddRobot
  - Simulation, [70](#)
- tryAddRobotTarget
  - Simulation, [70](#)
- tryAddStationaryLightSource
  - Simulation, [70](#)
- tryOpen
  - Simulation, [70](#)
- tryRemoveLightSource
  - Simulation, [70](#)
- tryRemoveObstacle
  - Simulation, [70](#)
- tryRemoveRobotTarget
  - Simulation, [70](#)
- trySave
  - Simulation, [70](#)
- update
  - PhysicalObject, [56](#)
  - Robot, [60](#)
- updateMembers
  - PhysicalObject, [57](#)
  - Robot, [60](#)
- updatePosition
  - PhysicalObject, [57](#)
  - Sensor, [64](#)
- util, [22](#)
  - addMovingLightSource, [23](#)
  - addNeuralNetworkRobotTarget, [23](#)
  - addObstacle, [24](#)
  - addRobot, [24](#)
  - addRobotTarget, [24](#)
  - addStationaryLightSource, [25](#)
  - advance, [25](#)
  - copy, [25](#)
  - display, [25](#)
  - getNumLights, [26](#)
  - getNumObstacles, [26](#)
  - getNumRobotsTargets, [26](#)
  - newColor, [26](#)
  - open, [26](#)
  - removeAllLightSource, [26](#)
  - removeAllObstacle, [26](#)
  - removeAllRobotTarget, [26](#)
  - removeLightSource, [26](#)
  - removeObstacle, [27](#)
  - removeRobotTarget, [27](#)
  - reset, [27](#)
  - save, [27](#)
- util.cpp, [91](#)
  - colorNum, [92](#)
  - lights, [92](#)
  - objectsMutex, [92](#)
  - obstacles, [92](#)
  - robots, [92](#)
  - targets, [92](#)
- util.h, [92](#)
- width
  - BaseGfxApp, [32](#)
- write
  - NeuralNetwork, [43](#)
- x
  - Location, [40](#)
- y
  - Location, [40](#)