ICP Project 2023/2024

1.0

Generated by Doxygen 1.9.8

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Ui Namespace Reference

Chapter 6

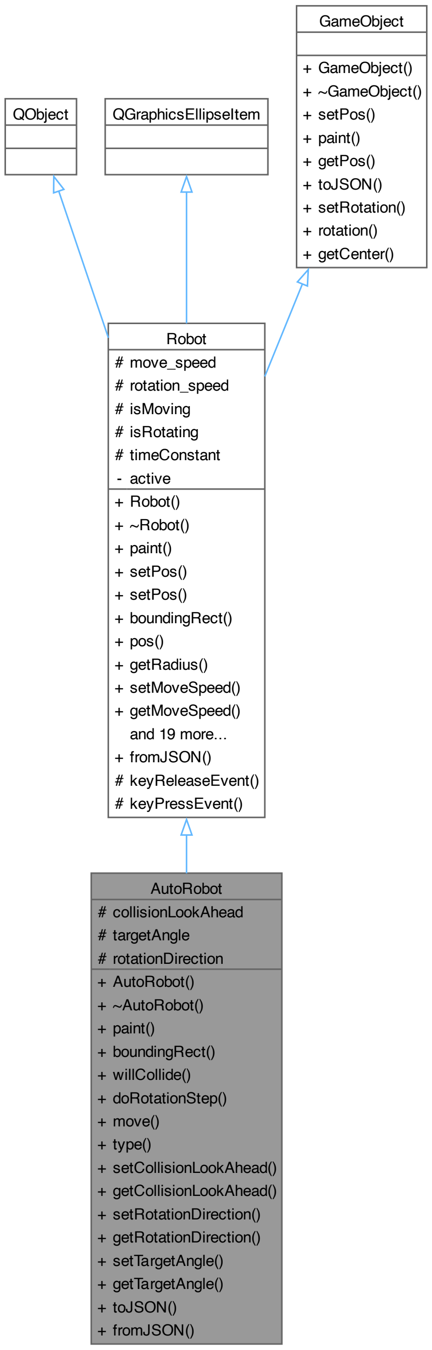
Class Documentation

6.1 AutoRobot Class Reference

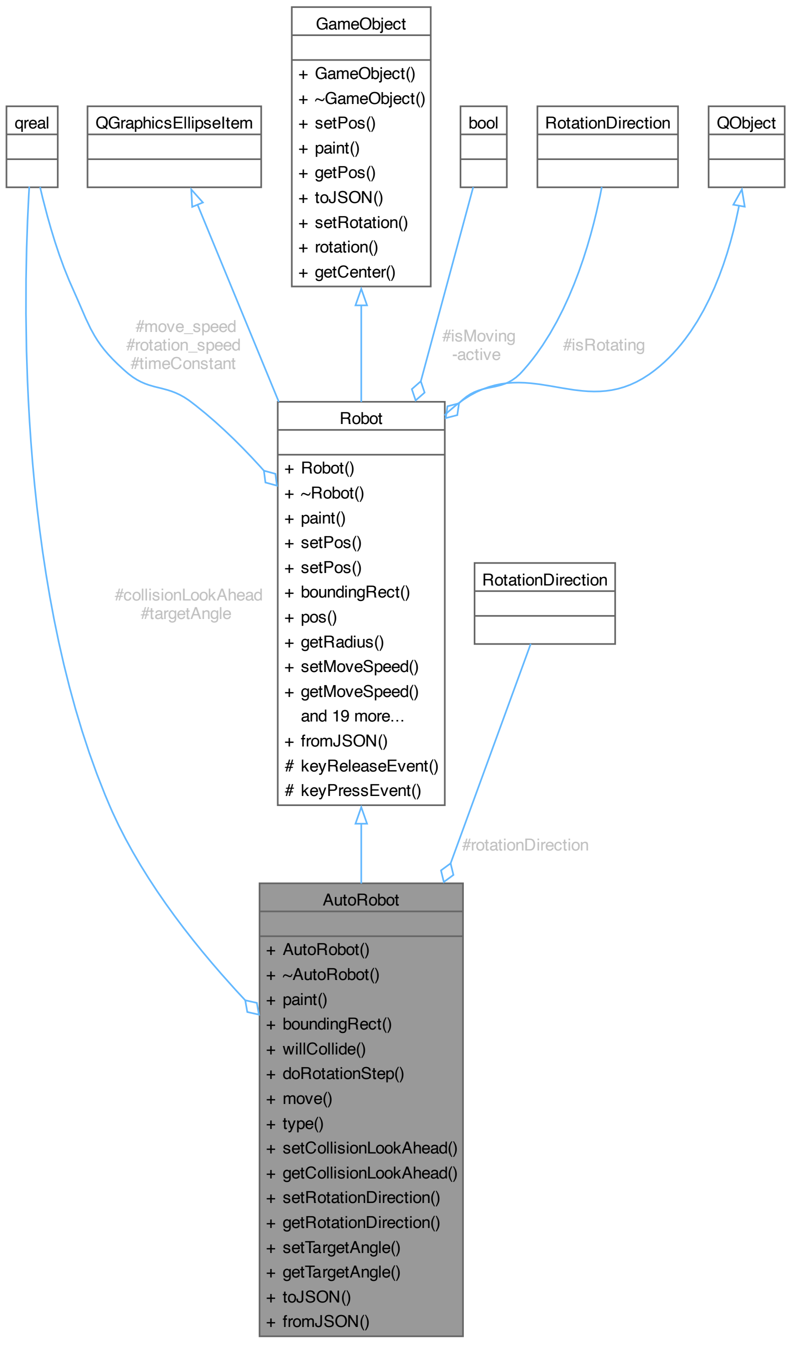
A class to represent an autonomous robot.

```
#include <autorobot.hpp>
```

Inheritance diagram for AutoRobot:



Collaboration diagram for AutoRobot:



Public Types

- enum { Type = QGraphicsItem::UserType + 2 }

Public Types inherited from Robot

- enum RotationDirection { Left = -1 , None = 0 , Right = 1 }

Enum to represent the direction of rotation of the robot.
- enum { Type = QGraphicsItem::UserType + 1 }

Public Member Functions

- AutoRobot (QGraphicsItem ∗parent=nullptr, qreal size=50, qreal collisionLookAhead=10, Robot::RotationDirection rotationDirection=Robot::RotationDirection::Right, qreal moveSpeed=1, qreal rotationSpeed=1, qreal ∗timeConstant=nullptr)

  Constructor for AutoRobot.
- ∼AutoRobot ()
- void paint (QPainter ∗painter, const QStyleOptionGraphicsItem ∗option, QWidget ∗widget) override
- QRectF boundingRect () const override
- bool willCollide (QPointF directionVector, qreal magnitude, bool allowAnticollision) override

  Check if the robot will collide with any object in the scene.
- void doRotationStep (RotationDirection direction)

  Perform a rotation step.
- bool move () override

  Perform a movement step.
- int type () const override

  Get the type of the object.
- void setCollisionLookAhead (qreal lookAhead)

  Set the look ahead distance for collision detection.
- qreal getCollisionLookAhead ()

  Get the look ahead distance for collision detection.
- void setRotationDirection (RotationDirection direction)

  Set the rotation direction of the robot.
- RotationDirection getRotationDirection ()

  Get the rotation direction of the robot.
- void setTargetAngle (qreal angle)

  Set the target angle of the robot.
- qreal getTargetAngle ()

  Get the target angle of the robot.
- QJsonObject toJSON () override

  Get the JSON representation of the object.

Public Member Functions inherited from Robot

- Robot (QGraphicsItem ∗parent=nullptr, qreal ∗timeConstant=nullptr)

  Default constructor.
- ∼Robot ()
- void setPos (const QPointF &pos)
- void setPos (qreal x, qreal y) override
- QPointF pos ()
- qreal getRadius () const
- void setMoveSpeed (qreal speed)

  Set the move speed of the robot.
- qreal getMoveSpeed ()

  Get the move speed of the robot.
- void setRotationSpeed (qreal speed)

  Set the rotation speed of the robot.

- qreal getRotationSpeed ()

  Get the rotation speed of the robot.
- void startMoving ()

  Allow the robot to be moved by setting the isMoving flag to true.
- void stopMoving ()

  Stop the robot from moving by setting the isMoving flag to false.
- void startRotating (RotationDirection direction)

  Start rotating the robot in the given direction.
- void stopRotating ()

  Stop the robot from rotating by setting the isRotating flag to None.
- QPointF getDirectionVector ()

  Get the direction vector of the robot.
- int type () const override

  Get the type of the robot.
- QPointF getPos () override

  Get the position of the robot.
- void toggleActive ()

  Toggle the active state of the robot.
- bool isActive ()

  Check if the robot is active.
- qreal getAngle ()

  Get the angle of the robot.
- void setRadius (qreal radius)

  Set the angle of the robot.
- QPointF getCenter () override

  Get the center of the robot.
- qreal rotation () override

  Get the time constant of the simulation.
- void setRotation (qreal angle) override

  Set the rotation of the robot.

Public Member Functions inherited from GameObject

- GameObject ()=default
- ∼GameObject ()=default

Static Public Member Functions

- static AutoRobot ∗ fromJSON (const QJsonObject &object, qreal ∗timeConstant)

  Create an AutoRobot object from a JSON object.

Static Public Member Functions inherited from Robot

- static Robot ∗ fromJSON (const QJsonObject &object, qreal ∗timeConstant)

  Create a Robot object from a JSON object.

Protected Attributes

- qreal collisionLookAhead = 0

  The look ahead distance for collision detection.
- qreal targetAngle = 0

  The target angle of the robot.
- Robot::RotationDirection rotationDirection = Robot::RotationDirection::Right

  The rotation direction of the robot.

Protected Attributes inherited from Robot

- qreal move_speed = 1

  The speed of the robot.
- qreal rotation_speed = 1

  The speed of the rotation of the robot.
- bool isMoving = false

  Flag to indicate if the robot is moving.
- RotationDirection isRotating = RotationDirection::None

  Flag to indicate the direction of rotation.
- qreal ∗ timeConstant = nullptr

  The time constant of the simulation.

Additional Inherited Members

Signals inherited from Robot

- void paramsUpdated ()

  Signal emitted when the parameters of the robot are updated.
- void robotSepuku ()

  Signal emitted when the robot is removed.

Protected Member Functions inherited from Robot

- void keyReleaseEvent (QKeyEvent ∗event)

  The radius of the robot.
- void keyPressEvent (QKeyEvent ∗event)

  Overridden keyPressEvent method.

6.1.1 Detailed Description

A class to represent an autonomous robot.

This class inherits from Robot and provides functionalities for an autonomous robot.

See also

Robot

Definition at line 23 of file autorobot.hpp.

6.1.2 Member Enumeration Documentation

6.1.2.1 anonymous enum

```
anonymous enum
```

Enumerator

Type	

Definition at line 27 of file autorobot.hpp.

```
00027 { Type = QGraphicsItem::UserType + 2 };
```

6.1.3 Constructor & Destructor Documentation

6.1.3.1 AutoRobot()

```
AutoRobot::AutoRobot (
            QGraphicsItem * parent = nullptr,
            qreal size = 50,
            qreal collisionLookAhead = 10,
            Robot::RotationDirection rotationDirection = Robot::RotationDirection::Right,
            qreal moveSpeed = 1,
            qreal rotationSpeed = 1,
            qreal * timeConstant = nullptr )
```

Constructor for AutoRobot.

Parameters

parent	The parent QGraphicsItem.
size	The size of the robot.
collisionLookAhead	The distance the robot looks ahead for collisions.
rotationDirection	The initial rotation direction of the robot.
moveSpeed	The movement speed of the robot.
rotationSpeed	The rotation speed of the robot.
timeConstant	A pointer to the time constant.

6.1.3.2 ∼AutoRobot()

```
AutoRobot::∼AutoRobot ( )
```

6.1.4 Member Function Documentation

6.1.4.1 boundingRect()

```
QRectF AutoRobot::boundingRect ( ) const  [override], [virtual]
```

Reimplemented from Robot.

6.1.4.2 doRotationStep()

```
void AutoRobot::doRotationStep (
            RotationDirection direction )
```

Perform a rotation step.

Parameters

direction	The direction of the rotation

Returns

void

6.1.4.3 fromJSON()

```
static AutoRobot * AutoRobot::fromJSON (
            const QJsonObject & object,
            qreal * timeConstant )  [static]
```

Create an AutoRobot object from a JSON object.

Parameters

object	The JSON object to create the AutoRobot object from
timeConstant	The time constant of the robot

Returns

AutoRobot∗ The AutoRobot object created from the JSON object

6.1.4.4 getCollisionLookAhead()

```
qreal AutoRobot::getCollisionLookAhead ( )  [inline]
```

Get the look ahead distance for collision detection.

Returns

qreal The look ahead distance

Definition at line 87 of file autorobot.hpp.

```
00087 { return collisionLookAhead; }
```

6.1.4.5 getRotationDirection()

```
RotationDirection AutoRobot::getRotationDirection ( )  [inline]
```

Get the rotation direction of the robot.

Returns

RotationDirection The rotation direction

Definition at line 100 of file autorobot.hpp.

```
00100 { return rotationDirection; }
```

6.1.4.6 getTargetAngle()

```
qreal AutoRobot::getTargetAngle ( )  [inline]
```

Get the target angle of the robot.

Returns

qreal The target angle

Definition at line 113 of file autorobot.hpp.

```
00113 { return targetAngle; }
```

6.1.4.7 move()

```
bool AutoRobot::move ( )  [override], [virtual]
```

Perform a movement step.

Returns

bool Whether the movement step was successful

Reimplemented from Robot.

6.1.4.8 paint()

```
void AutoRobot::paint (
             QPainter * painter,
             const QStyleOptionGraphicsItem * option,
             QWidget * widget )  [override], [virtual]
```

Override the paint method to draw a line showing the direction of the robot

Reimplemented from Robot.

6.1.4.9 setCollisionLookAhead()

```
void AutoRobot::setCollisionLookAhead (
             qreal lookAhead )  [inline]
```

Set the look ahead distance for collision detection.

Parameters

lookAhead	The look ahead distance

Returns

void

Definition at line 81 of file autorobot.hpp.

```
00081 { collisionLookAhead = lookAhead; }
```

6.1.4.10 setRotationDirection()

```
void AutoRobot::setRotationDirection (
            RotationDirection direction )  [inline]
```

Set the rotation direction of the robot.

Parameters

direction	The rotation direction

Returns

void

Definition at line 94 of file autorobot.hpp.

```
00094 { rotationDirection = direction; }
```

6.1.4.11 setTargetAngle()

```
void AutoRobot::setTargetAngle (
            qreal angle )  [inline]
```

Set the target angle of the robot.

Parameters

angle	The target angle

Returns

void

Definition at line 107 of file autorobot.hpp.

```
00107 { targetAngle = angle; }
```

6.1.4.12 toJSON()

```
QJsonObject AutoRobot::toJSON ( )  [override], [virtual]
```

Get the JSON representation of the object.

Returns

QJsonObject The JSON representation of the object

Reimplemented from Robot.

6.1.4.13 type()

```
int AutoRobot::type ( ) const  [inline], [override]
```

Get the type of the object.

Returns

int The type of the object

Definition at line 74 of file autorobot.hpp.

```
00074 { return Type; }
```

6.1.4.14 willCollide()

```
bool AutoRobot::willCollide (
             QPointF directionVector,
             qreal magnitude,
             bool allowAnticollision )  [override], [virtual]
```

Check if the robot will collide with any object in the scene.

Parameters

directionVector	The direction vector of the robot
magnitude	The magnitude of the direction vector
allowAnticollision	Whether to allow anticollision

Returns

bool Whether the robot will collide with any object in the scene

Reimplemented from Robot.

6.1.5 Member Data Documentation

6.1.5.1 collisionLookAhead

```
qreal AutoRobot::collisionLookAhead = 0  [protected]
```

The look ahead distance for collision detection.

Definition at line 131 of file autorobot.hpp.

6.1.5.2 rotationDirection

```
Robot::RotationDirection AutoRobot::rotationDirection = Robot::RotationDirection::Right  [protected]
```

The rotation direction of the robot.

Definition at line 137 of file autorobot.hpp.

6.1.5.3 targetAngle

```
qreal AutoRobot::targetAngle = 0 [protected]
```

The target angle of the robot.

Definition at line 134 of file autorobot.hpp.

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

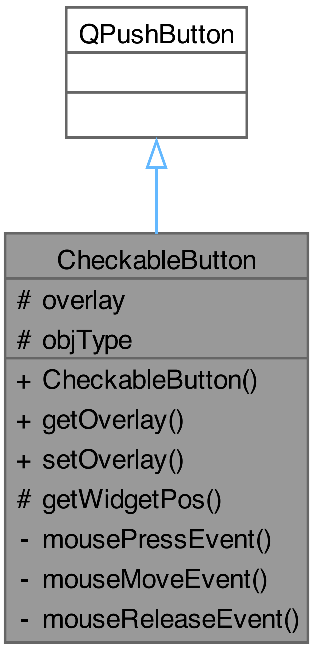
- autorobot.hpp

6.2 CheckableButton Class Reference

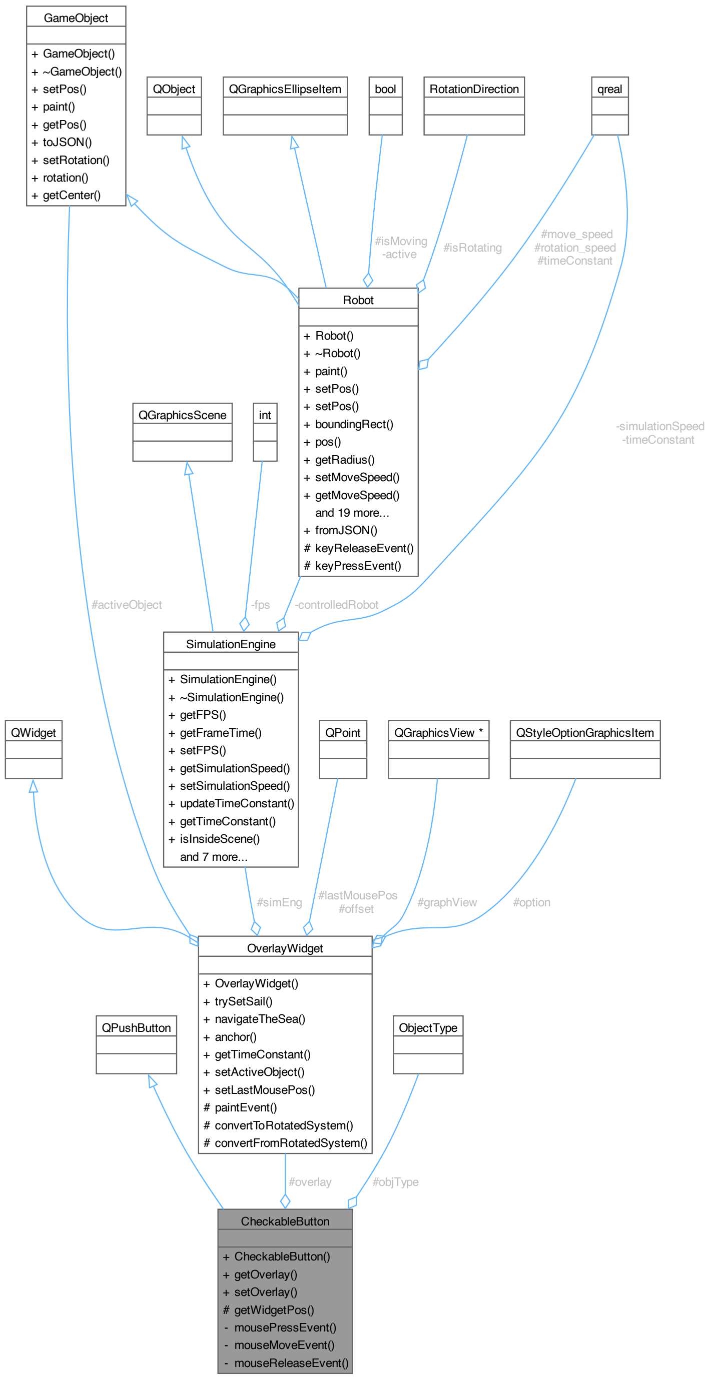
A class to represent a checkable button.

```
#include <checkablebutton.hpp>
```

Inheritance diagram for CheckableButton:



Collaboration diagram for CheckableButton:



Public Types

- enum ObjectType { AUTO , CONT , OBST }

  Enum to represent the type of object that the button represents AUTO: AutoRobot CONT: ControlledRobot OBST: Obstacle.

Public Member Functions

- CheckableButton (const QString &text, QWidget ∗parent=nullptr, ObjectType type=ObjectType::OBST)

    Constructor for CheckableButton.
- OverlayWidget ∗ getOverlay () const

    Get the overlay widget of the button.
- void setOverlay (OverlayWidget ∗overlay)

    Set the overlay widget of the button.

Protected Member Functions

- QPoint getWidgetPos (QPoint localPos)

    Get the position of the widget on the grid.

Protected Attributes

- OverlayWidget ∗ overlay

    Pointer to the overlay widget.
- ObjectType objType

    The type of object that the button represents.

Private Slots

- void mousePressEvent (QMouseEvent ∗event) override

    Override the mousePressEvent method.
- void mouseMoveEvent (QMouseEvent ∗event) override

    Override the mouseMoveEvent method.
- void mouseReleaseEvent (QMouseEvent ∗event) override

    Override the mouseReleaseEvent method.

6.2.1 Detailed Description

A class to represent a checkable button.

This class inherits from QPushButton and provides functionalities for a button that can be checked and unchecked. It also has an OverlayWidget that is used to draw the object on the grid.

See also

QPushButton

Definition at line 24 of file checkablebutton.hpp.

6.2.2 Member Enumeration Documentation

6.2.2.1 ObjectType

`enum CheckableButton::ObjectType`

Enum to represent the type of object that the button represents AUTO: AutoRobot CONT: ControlledRobot OBST: Obstacle.

-

Enumerator

AUTO	
CONT	
OBST	

Definition at line 31 of file checkablebutton.hpp.

```
00031                    {
00032         AUTO,
00033         CONT,
00034         OBST
00035     };
```

6.2.3 Constructor & Destructor Documentation

6.2.3.1 CheckableButton()

```
CheckableButton::CheckableButton (
            const QString & text,
            QWidget * parent = nullptr,
            ObjectType type = ObjectType::OBST )  [explicit]
```

Constructor for CheckableButton.

Parameters

text	The text to be displayed on the button.
parent	The parent QWidget.
type	The type of object that the button represents.

6.2.4 Member Function Documentation

6.2.4.1 getOverlay()

```
OverlayWidget * CheckableButton::getOverlay ( ) const  [inline]
```

Get the overlay widget of the button.

Returns

OverlayWidget∗ The overlay widget of the button

Definition at line 49 of file checkablebutton.hpp.

```
00049 { return overlay; }
```

6.2.4.2 getWidgetPos()

```
QPoint CheckableButton::getWidgetPos (
            QPoint localPos )  [protected]
```

Get the position of the widget on the grid.

Parameters

localPos	The local position of the mouse.

Returns

QPoint The position in the overlay widget.

6.2.4.3 mouseMoveEvent

```
void CheckableButton::mouseMoveEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Override the mouseMoveEvent method.

Parameters

event	The mouse event

Returns

void

6.2.4.4 mousePressEvent

```
void CheckableButton::mousePressEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Override the mousePressEvent method.

Parameters

event	The mouse event

Returns

void

6.2.4.5 mouseReleaseEvent

```
void CheckableButton::mouseReleaseEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Override the mouseReleaseEvent method.

Parameters

event	The mouse event

Returns

void

6.2.4.6 setOverlay()

```
void CheckableButton::setOverlay (
             OverlayWidget * overlay )  [inline]
```

Set the overlay widget of the button.

Parameters

overlay	The overlay widget to set

Returns

void

Definition at line 56 of file checkablebutton.hpp.

```
00056 { this->overlay = overlay; }
```

6.2.5 Member Data Documentation

6.2.5.1 objType

```
ObjectType CheckableButton::objType  [protected]
```

The type of object that the button represents.

Definition at line 63 of file checkablebutton.hpp.

6.2.5.2 overlay

```
OverlayWidget* CheckableButton::overlay  [protected]
```

Pointer to the overlay widget.

Definition at line 60 of file checkablebutton.hpp.

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

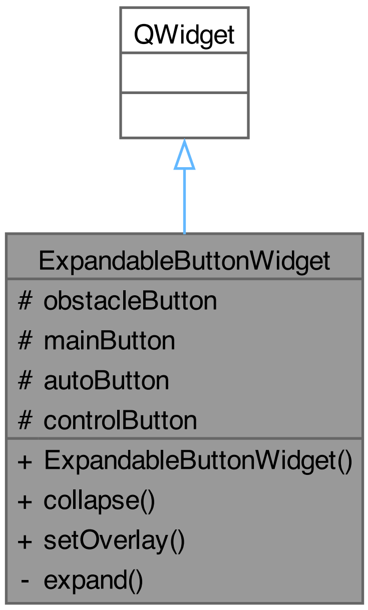
- checkablebutton.hpp

6.3 ExpandableButtonWidget Class Reference

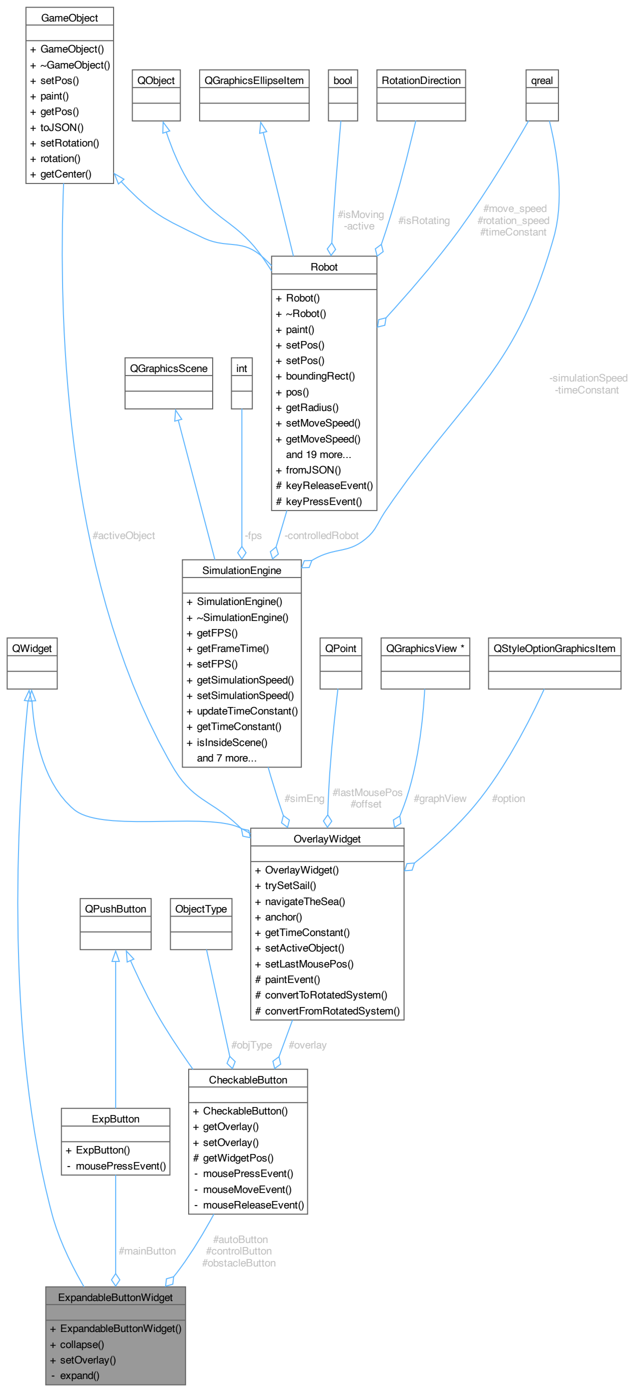
A class to represent an expandable button widget.

```
#include <expbuttonwidget.hpp>
```

Inheritance diagram for ExpandableButtonWidget:



Collaboration diagram for ExpandableButtonWidget:



Public Member Functions

- ExpandableButtonWidget (QWidget ∗parent=nullptr)

  Construct a new Expandable Button Widget object.
- void collapse ()

  Get the obstacle button.
- void setOverlay (OverlayWidget ∗overlay)

  Get the obstacle button.

Protected Attributes

- CheckableButton ∗ obstacleButton

  Reference to the obstacle button.
- ExpButton ∗ mainButton

  Reference to the main button.
- CheckableButton ∗ autoButton

  Reference to the auto button.
- CheckableButton ∗ controlButton

  Reference to the control button.

Private Slots

- void expand ()

  Slot to handle the main button press event.

6.3.1 Detailed Description

A class to represent an expandable button widget.

This class provides an interface for creating and managing expandable button widgets.

See also

QWidget

Definition at line 30 of file expbuttonwidget.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 ExpandableButtonWidget()

```
ExpandableButtonWidget::ExpandableButtonWidget (
             QWidget * parent = nullptr )  [explicit]
```

Construct a new Expandable Button Widget object.

Parameters

parent	The parent widget. Default is nullptr.

6.3.3 Member Function Documentation

6.3.3.1 collapse()

```
void ExpandableButtonWidget::collapse ( )
```

Get the obstacle button.

Returns

CheckableButton* The obstacle button.

6.3.3.2 expand

```
void ExpandableButtonWidget::expand ( )  [private], [slot]
```

Slot to handle the main button press event.

Returns

void

6.3.3.3 setOverlay()

```
void ExpandableButtonWidget::setOverlay (
            OverlayWidget * overlay )
```

Get the obstacle button.

Returns

CheckableButton* The obstacle button.

6.3.4 Member Data Documentation

6.3.4.1 autoButton

```
CheckableButton* ExpandableButtonWidget::autoButton  [protected]
```

Reference to the auto button.

Definition at line 60 of file expbuttonwidget.hpp.

6.3.4.2 controlButton

```
CheckableButton* ExpandableButtonWidget::controlButton  [protected]
```

Reference to the control button.

Definition at line 63 of file expbuttonwidget.hpp.

6.3.4.3 mainButton

```
ExpButton* ExpandableButtonWidget::mainButton  [protected]
```

Reference to the main button.

Definition at line 57 of file expbuttonwidget.hpp.

6.3.4.4 obstacleButton

```
CheckableButton* ExpandableButtonWidget::obstacleButton  [protected]
```

Reference to the obstacle button.

Definition at line 54 of file expbuttonwidget.hpp.

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

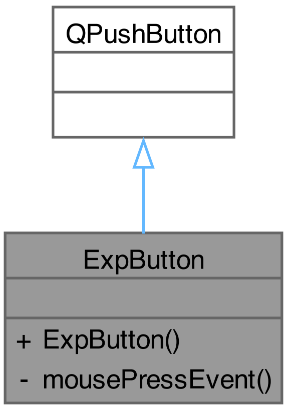
- expbuttonwidget.hpp

6.4 ExpButton Class Reference

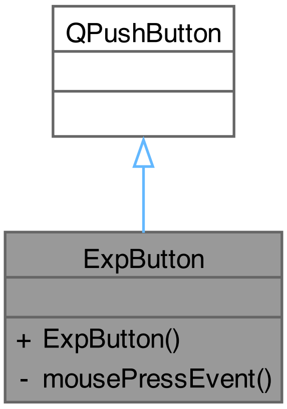
A class for expandable buttons.

```
#include <expbutton.hpp>
```

Inheritance diagram for ExpButton:



Collaboration diagram for ExpButton:



Signals

- void pressed ()

  Signal emitted when the button is pressed.

Public Member Functions

- ExpButton (const QString &text, QWidget ∗parent=nullptr)

  Constructor for ExpButton.

Private Slots

- void mousePressEvent (QMouseEvent ∗event) override

  Slot to handle the button press event.

6.4.1 Detailed Description

A class for expandable buttons.

This class inherits from QPushButton and emits a signal when pressed.

See also

QPushButton

Definition at line 21 of file expbutton.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ExpButton()

```
ExpButton::ExpButton (
            const QString & text,
            QWidget * parent = nullptr )  [explicit]
```

Constructor for ExpButton.

Parameters

text	The text to be displayed on the button.
parent	The parent QWidget.

6.4.3 Member Function Documentation

6.4.3.1 mousePressEvent

```
void ExpButton::mousePressEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Slot to handle the button press event.

Parameters

event	The QMouseEvent that triggered the slot.

Returns

void

6.4.3.2 pressed

```
void ExpButton::pressed ( )  [signal]
```

Signal emitted when the button is pressed.

Returns

void

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

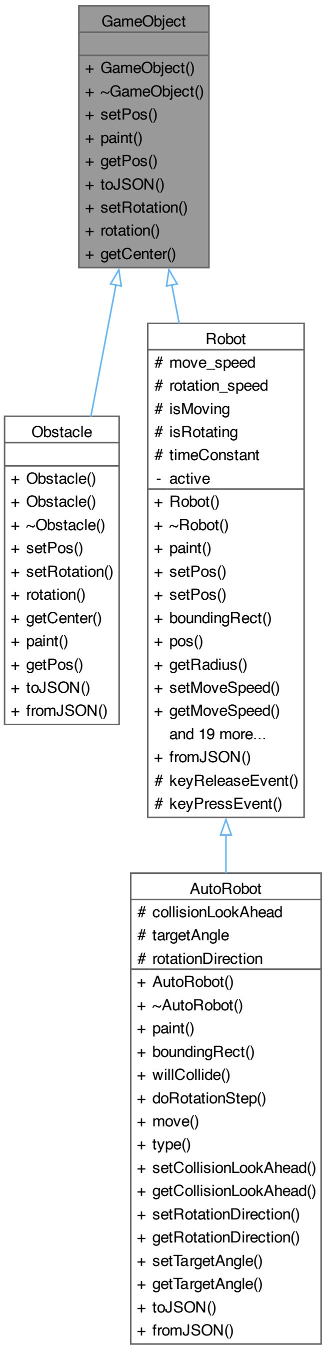
- expbutton.hpp

6.5 GameObject Class Reference

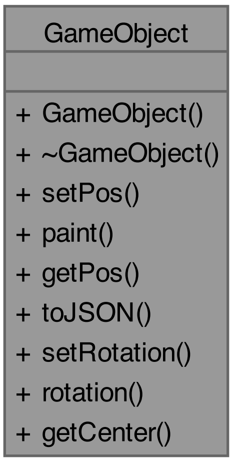
A class to represent a game object in the simulation.

```
#include <gameobject.hpp>
```

Inheritance diagram for GameObject:



Collaboration diagram for GameObject:



Public Member Functions

- GameObject ()=default
- ∼GameObject ()=default
- virtual void setPos (qreal x, qreal y)=0

  Set the position of the game object.
- virtual void paint (QPainter ∗painter, const QStyleOptionGraphicsItem ∗option, QWidget ∗widget)=0

  Paint the game object.
- virtual QPointF getPos ()=0

  Get the position of the game object.
- virtual QJsonObject toJSON ()=0

  Convert the game object to a JSON object.
- virtual void setRotation (qreal angle)=0

  Set the rotation of the game object.
- virtual qreal rotation ()=0

  Get the rotation of the game object.
- virtual QPointF getCenter ()=0

  Get the center of the game object.

6.5.1 Detailed Description

A class to represent a game object in the simulation.

This class provides an interface for creating and managing game objects.

Definition at line 20 of file gameobject.hpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 GameObject()

```
GameObject::GameObject ( )  [default]
```

6.5.2.2 ∼GameObject()

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

6.5.3 Member Function Documentation

6.5.3.1 getCenter()

```
virtual QPointF GameObject::getCenter ( )  [pure virtual]
```

Get the center of the game object.

Returns

QPointF

Implemented in Obstacle, and Robot.

6.5.3.2 getPos()

```
virtual QPointF GameObject::getPos ( )  [pure virtual]
```

Get the position of the game object.

Returns

QPointF

Implemented in Obstacle, and Robot.

6.5.3.3 paint()

```
virtual void GameObject::paint (
            QPainter ∗ painter,
            const QStyleOptionGraphicsItem ∗ option,
            QWidget ∗ widget )  [pure virtual]
```

Paint the game object.

Parameters

painter	
option	
widget	

Returns

void

Implemented in AutoRobot, Obstacle, and Robot.

6.5.3.4 rotation()

```
virtual qreal GameObject::rotation ( )  [pure virtual]
```

Get the rotation of the game object.

Returns

qreal

Implemented in Obstacle, and Robot.

6.5.3.5 setPos()

```
virtual void GameObject::setPos (
             qreal x,
             qreal y )  [pure virtual]
```

Set the position of the game object.

Parameters

x	
y	

Returns

void

Implemented in Obstacle, and Robot.

6.5.3.6 setRotation()

```
virtual void GameObject::setRotation (
             qreal angle )  [pure virtual]
```

Set the rotation of the game object.

Parameters

angle	

Returns

void

Implemented in Obstacle, and Robot.

6.5.3.7 toJSON()

```
virtual QJsonObject GameObject::toJSON ( )  [pure virtual]
```

Convert the game object to a JSON object.

Returns

QJsonObject

Implemented in AutoRobot, Obstacle, and Robot.

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

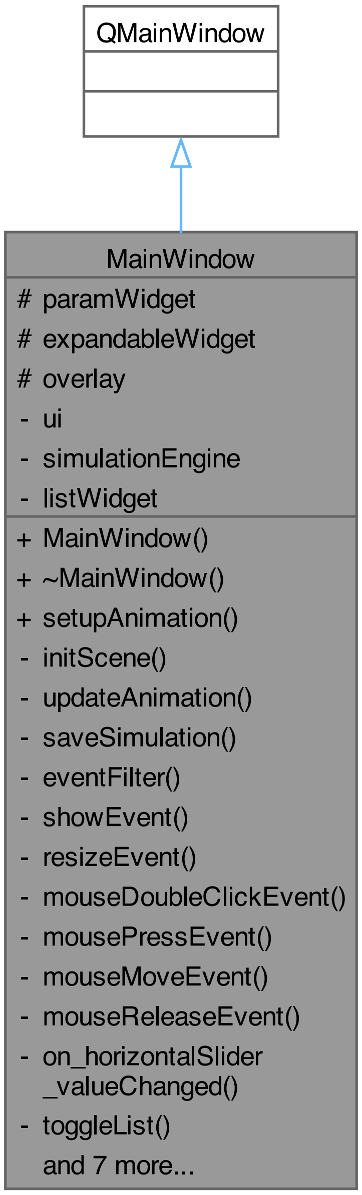
- gameobject.hpp

6.6 MainWindow Class Reference

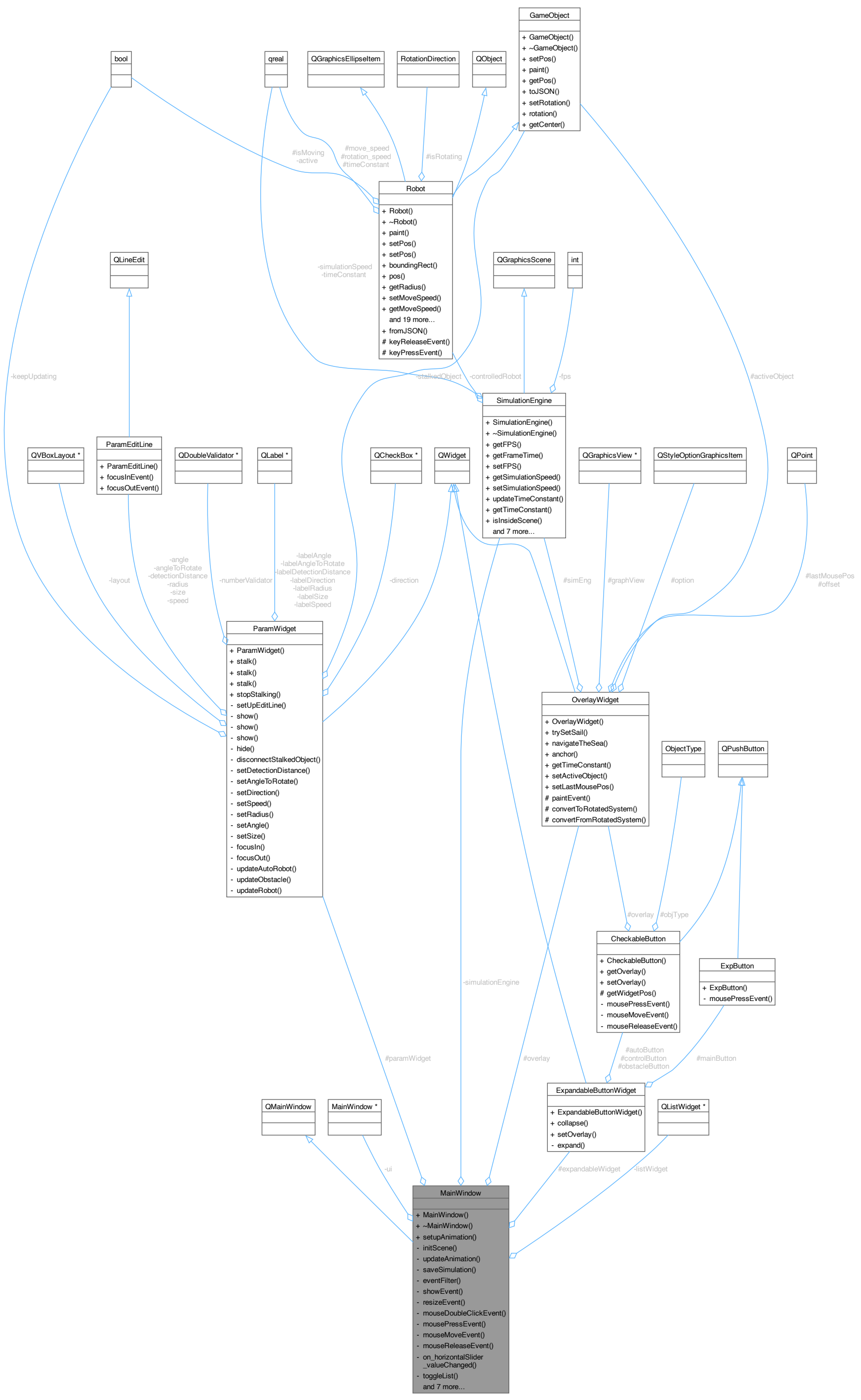
A class to represent the main window of the application.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)
- ∼MainWindow ()
- void setupAnimation ()

Protected Attributes

- ParamWidget ∗ paramWidget

  The param widget.
- ExpandableButtonWidget ∗ expandableWidget

  The expandable button widget.
- OverlayWidget ∗ overlay

  The overlay widget.

Private Slots

- void saveSimulation ()

  Slot to handle the save button click event.
- bool eventFilter (QObject ∗object, QEvent ∗event) override

  Overridden event filter method to handle key press events.
- void showEvent (QShowEvent ∗event) override

  Overriden show event method to handle the show event.
- void resizeEvent (QResizeEvent ∗event) override

  Overriden resize event method to handle the resize event.
- void mouseDoubleClickEvent (QMouseEvent ∗event) override

  Overriden close event method to handle the close event.
- void mousePressEvent (QMouseEvent ∗event) override

  Overriden mouse press event method to handle the mouse press event.
- void mouseMoveEvent (QMouseEvent ∗event) override

  Overriden mouse move event method to handle the mouse move event.
- void mouseReleaseEvent (QMouseEvent ∗event) override

  Overriden mouse release event method to handle the mouse release event.
- void on_horizontalSlider_valueChanged (int value)

  Slot to handle the horizontal slider value changed event.
- void toggleList ()

  Slot to handle toggling the list.
- void handleItemDoubleClick (QListWidgetItem ∗item)

  Slot to handle the item double click event from the list.
- void on_pushButton_clicked ()

  Slot to handle clear button click event.
- void goLeft ()

  Slot to handle rotate anticlockwise button click event.
- void stopRotating ()

  Slot to handle stop rotating button click event.
- void goRight ()

  Slot to handle rotate clockwise button click event.
- void goStraight ()

  Slot to handle move forward button click event.
- void stopMoving ()

  Slot to handle stop moving button click event.

Private Member Functions

- void initScene ()
- void updateAnimation ()

Private Attributes

- Ui::MainWindow ∗ ui

    The UI object.
- SimulationEngine ∗ simulationEngine

    The simulation engine.
- QListWidget ∗ listWidget

    The list widget.

6.6.1 Detailed Description

A class to represent the main window of the application.

This class inherits from QMainWindow and provides the main window of the application.

See also

QMainWindow

Definition at line 43 of file mainwindow.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 MainWindow()

```
MainWindow::MainWindow (
             QWidget ∗ parent = nullptr )
```

6.6.2.2 ∼MainWindow()

```
MainWindow::∼MainWindow ( )
```

6.6.3 Member Function Documentation

6.6.3.1 eventFilter

```
bool MainWindow::eventFilter (
             QObject ∗ object,
             QEvent ∗ event )  [override], [private], [slot]
```

Overridden event filter method to handle key press events.

Parameters

object	The object that the event is being filtered for
event	The event that is being filtered

Returns

bool Whether the event was handled

6.6.3.2 goLeft

```
void MainWindow::goLeft ( )  [private], [slot]
```

Slot to handle rotate anticlockwise button click event.

Returns

void

6.6.3.3 goRight

```
void MainWindow::goRight ( )  [private], [slot]
```

Slot to handle rotate clockwise button click event.

Returns

void

6.6.3.4 goStraight

```
void MainWindow::goStraight ( )  [private], [slot]
```

Slot to handle move forward button click event.

Returns

void

6.6.3.5 handleItemDoubleClick

```
void MainWindow::handleItemDoubleClick (
            QListWidgetItem * item )  [private], [slot]
```

Slot to handle the item double click event from the list.

Parameters

item	The item that was double clicked

Returns

void

6.6.3.6 initScene()

```
void MainWindow::initScene ( )  [private]
```

6.6.3.7 mouseDoubleClickEvent

```
void MainWindow::mouseDoubleClickEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Overriden close event method to handle the close event.

Parameters

event	The close event

Returns

void

6.6.3.8 mouseMoveEvent

```
void MainWindow::mouseMoveEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Overriden mouse move event method to handle the mouse move event.

Parameters

event	The mouse move event

Returns

void

6.6.3.9 mousePressEvent

```
void MainWindow::mousePressEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Overriden mouse press event method to handle the mouse press event.

Parameters

event	The mouse press event

Returns

void

6.6.3.10 mouseReleaseEvent

```
void MainWindow::mouseReleaseEvent (
            QMouseEvent * event )  [override], [private], [slot]
```

Overriden mouse release event method to handle the mouse release event.

Parameters

event	The mouse release event

Returns

void

6.6.3.11 on_horizontalSlider_valueChanged

```
void MainWindow::on_horizontalSlider_valueChanged (
            int value )  [private], [slot]
```

Slot to handle the horizontal slider value changed event.

Parameters

value	The new value of the slider

Returns

void

6.6.3.12 on_pushButton_clicked

```
void MainWindow::on_pushButton_clicked ( )  [private], [slot]
```

Slot to handle clear button click event.

Returns

void

6.6.3.13 resizeEvent

```
void MainWindow::resizeEvent (
            QResizeEvent * event )  [override], [private], [slot]
```

Overriden resize event method to handle the resize event.

Parameters

event	The resize event

Returns

void

6.6.3.14 saveSimulation

```
void MainWindow::saveSimulation ( )  [private], [slot]
```

Slot to handle the save button click event.

Returns

void

6.6.3.15 setupAnimation()

```
void MainWindow::setupAnimation ( )
```

6.6.3.16 showEvent

```
void MainWindow::showEvent (
            QShowEvent * event )  [override], [private], [slot]
```

Overriden show event method to handle the show event.

Parameters

event	The show event

Returns

void

6.6.3.17 stopMoving

```
void MainWindow::stopMoving ( )  [private], [slot]
```

Slot to handle stop moving button click event.

Returns

void

6.6.3.18 stopRotating

```
void MainWindow::stopRotating ( )  [private], [slot]
```

Slot to handle stop rotating button click event.

Returns

void

6.6.3.19 toggleList

```
void MainWindow::toggleList ( )  [private], [slot]
```

Slot to handle toggling the list.

Returns

void

6.6.3.20 updateAnimation()

```
void MainWindow::updateAnimation ( )  [private]
```

6.6.4 Member Data Documentation

6.6.4.1 expandableWidget

```
ExpandableButtonWidget* MainWindow::expandableWidget  [protected]
```

The expandable button widget.

Definition at line 69 of file mainwindow.h.

6.6.4.2 listWidget

```
QListWidget* MainWindow::listWidget  [private]
```

The list widget.

Definition at line 59 of file mainwindow.h.

6.6.4.3 overlay

`OverlayWidget* MainWindow::overlay [protected]`

The overlay widget.

Definition at line 72 of file mainwindow.h.

6.6.4.4 paramWidget

`ParamWidget* MainWindow::paramWidget [protected]`

The param widget.

Definition at line 66 of file mainwindow.h.

6.6.4.5 simulationEngine

`SimulationEngine* MainWindow::simulationEngine [private]`

The simulation engine.

Definition at line 56 of file mainwindow.h.

6.6.4.6 ui

`Ui::MainWindow* MainWindow::ui [private]`

The UI object.

Definition at line 53 of file mainwindow.h.

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

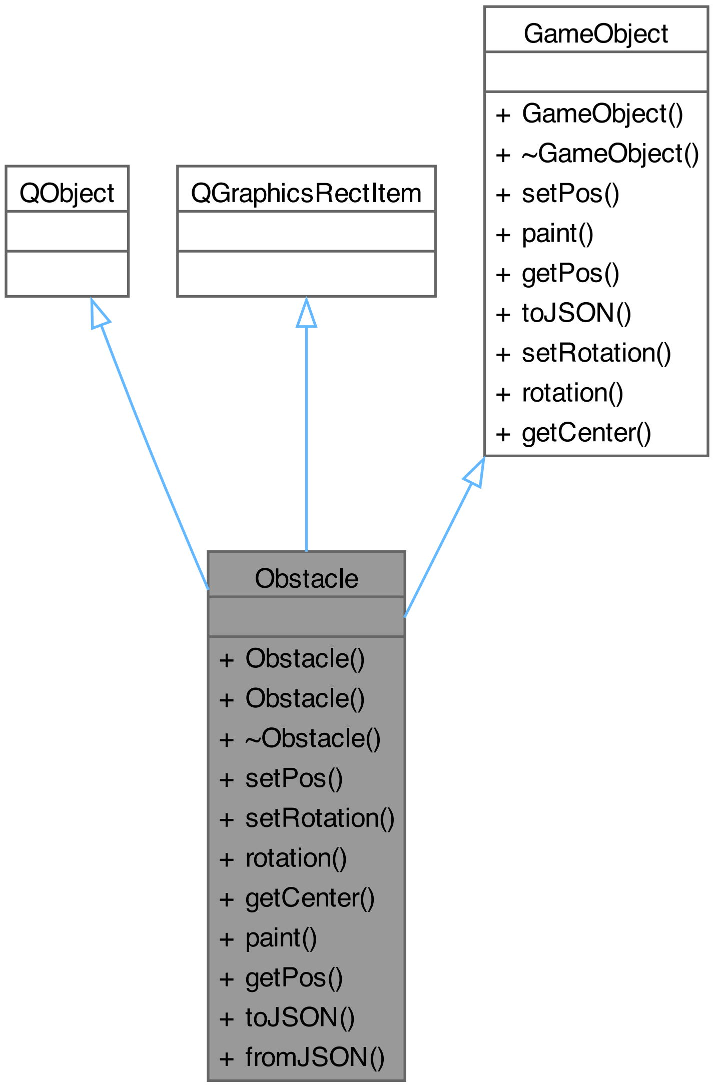
- mainwindow.h

6.7 Obstacle Class Reference

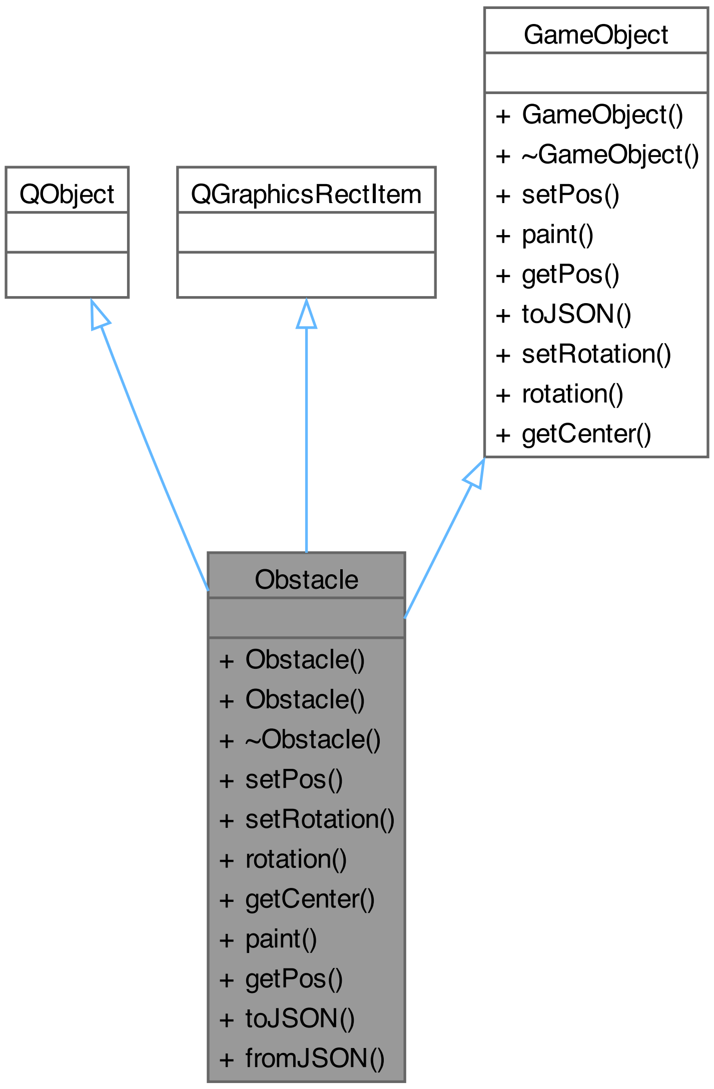
A class to represent an obstacle.

```
#include <obstacle.hpp>
```

Inheritance diagram for Obstacle:



Collaboration diagram for Obstacle:



Signals

- void paramsUpdated ()

  Signal emitted when the parameters of the obstacle are updated.
- void obstacleSepuku ()

  Signal emitted when the obstacle is removed.

Public Member Functions

- Obstacle (QGraphicsItem ∗parent=nullptr)

  Default constructor.

- Obstacle (const Obstacle &)

  Copy constructor.
- ∼Obstacle ()

  Destructor.
- void setPos (qreal x, qreal y) override

  Set the position of the obstacle.
- void setRotation (qreal angle) override

  Set the rotation of the obstacle.
- qreal rotation () override

  Get the rotation of the obstacle.
- QPointF getCenter () override

  Get the center of the obstacle.
- void paint (QPainter ∗painter, const QStyleOptionGraphicsItem ∗option, QWidget ∗widget) override

  Paint the obstacle.
- QPointF getPos () override

  Get the position of the obstacle.
- QJsonObject toJSON () override

  Convert the obstacle to a JSON object.

Public Member Functions inherited from GameObject

- GameObject ()=default
- ∼GameObject ()=default

Static Public Member Functions

- static Obstacle ∗ fromJSON (const QJsonObject &json)

  Create an Obstacle object from a JSON object.

6.7.1 Detailed Description

A class to represent an obstacle.

This class inherits from QGraphicsRectItem and GameObject. It represents an obstacle in a game.

Definition at line 23 of file obstacle.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Obstacle() [1/2]

```
Obstacle::Obstacle (
             QGraphicsItem * parent = nullptr )
```

Default constructor.

Parameters

parent	The parent QGraphicsItem.

Returns

void

6.7.2.2 Obstacle() [2/2]

```
Obstacle::Obstacle (
            const Obstacle &  )  [inline]
```

Copy constructor.

Parameters

Obstacle	The Obstacle object to copy.

Returns

void

Definition at line 39 of file obstacle.hpp.

```
00040         : QGraphicsRectItem() {}
```

6.7.2.3 ∼Obstacle()

```
Obstacle::∼Obstacle ( )
```

Destructor.

6.7.3 Member Function Documentation

6.7.3.1 fromJSON()

```
static Obstacle * Obstacle::fromJSON (
            const QJsonObject & json )  [static]
```

Create an Obstacle object from a JSON object.

Parameters

json	The QJsonObject to convert.

Returns

A pointer to the created Obstacle object.

6.7.3.2 getCenter()

```
QPointF Obstacle::getCenter ( )  [inline], [override], [virtual]
```

Get the center of the obstacle.

Returns

The center of the obstacle as a QPointF.

Implements GameObject.

Definition at line 72 of file obstacle.hpp.

```
00072 { return boundingRect().center(); }
```

6.7.3.3 getPos()

```
QPointF Obstacle::getPos ( )  [override], [virtual]
```

Get the position of the obstacle.

Returns

The position of the obstacle as a QPointF object.

Implements GameObject.

6.7.3.4 obstacleSepuku

```
void Obstacle::obstacleSepuku ( )  [signal]
```

Signal emitted when the obstacle is removed.

Returns

void

6.7.3.5 paint()

```
void Obstacle::paint (
             QPainter * painter,
             const QStyleOptionGraphicsItem * option,
             QWidget * widget )  [override], [virtual]
```

Paint the obstacle.

Parameters

painter	Pointer to the QPainter object.
option	Pointer to the QStyleOptionGraphicsItem object.
widget	Pointer to the QWidget object.

Implements GameObject.

6.7.3.6 paramsUpdated

```
void Obstacle::paramsUpdated ( )  [signal]
```

Signal emitted when the parameters of the obstacle are updated.

Returns

void

6.7.3.7 rotation()

```
qreal Obstacle::rotation ( )  [inline], [override], [virtual]
```

Get the rotation of the obstacle.

Returns

The rotation of the obstacle as a qreal.

Implements GameObject.

Definition at line 66 of file obstacle.hpp.

```
00066 { return QGraphicsRectItem::rotation(); }
```

6.7.3.8 setPos()

```
void Obstacle::setPos (
             qreal x,
             qreal y )  [override], [virtual]
```

Set the position of the obstacle.

Parameters

x	The x-coordinate of the position.
y	The y-coordinate of the position.

Returns

void

Implements GameObject.

6.7.3.9 setRotation()

```
void Obstacle::setRotation (
             qreal angle )  [override], [virtual]
```

Set the rotation of the obstacle.

Parameters

angle	The angle of the rotation.

Returns

void

Implements GameObject.

6.7.3.10 toJSON()

```
QJsonObject Obstacle::toJSON ( )  [override], [virtual]
```

Convert the obstacle to a JSON object.

Returns

The obstacle as a QJsonObject.

Implements GameObject.

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

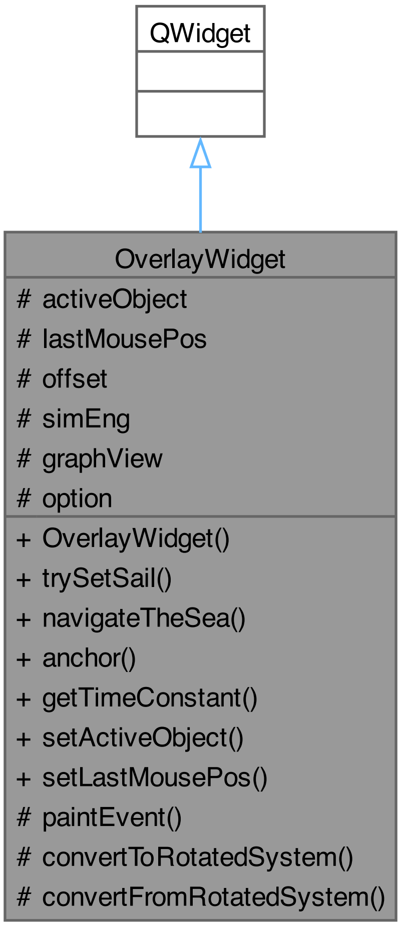
- obstacle.hpp

6.8 OverlayWidget Class Reference

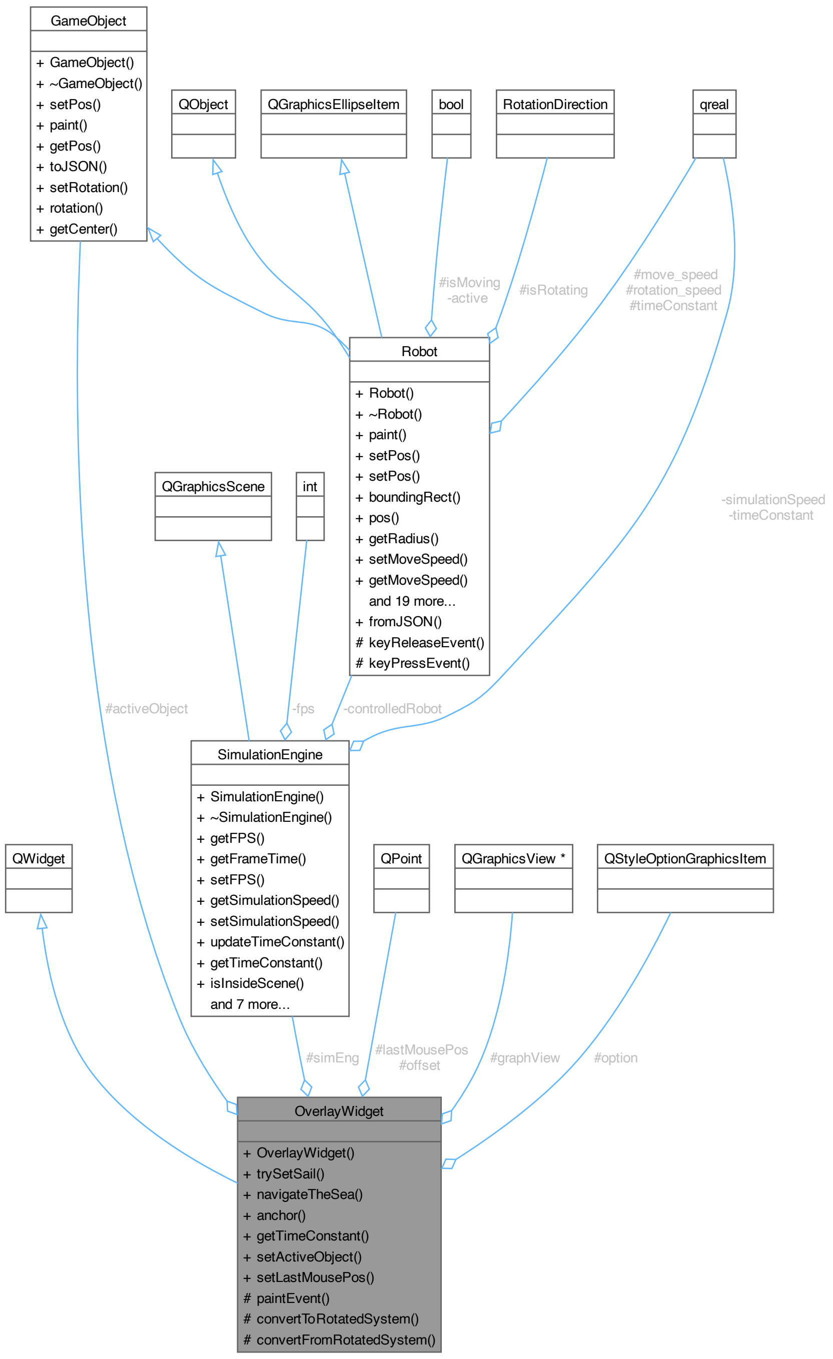
A class to represent an overlay widget.

```
#include <overlaywidget.hpp>
```

Inheritance diagram for OverlayWidget:



Collaboration diagram for OverlayWidget:



Public Member Functions

- OverlayWidget (QWidget ∗parent=nullptr, SimulationEngine ∗simEng=nullptr, QGraphicsView ∗graphView=nullptr)

  Construct a new Overlay Widget object.
- void trySetSail (QMouseEvent ∗event)

  Try grab the object based on the mouse position.
- void navigateTheSea (QMouseEvent ∗event)

Drag the object based on the mouse position in the overlay.
- void anchor ()

  Anchor the object based on the mouse position back to scene.
- qreal ∗ getTimeConstant ()

  Get the time constant of the simulation engine.
- void setActiveObject (GameObject ∗obj)

  Get the active object.
- void setLastMousePos (QPoint pos)

  Get the last mouse position.

Protected Member Functions

- void paintEvent (QPaintEvent ∗event) override

  Override the mousePressEvent method.
- QPoint convertToRotatedSystem (QPoint point, qreal angle)

  Convert the point to the rotated system.
- QPoint convertFromRotatedSystem (QPoint point, qreal angle)

  Convert the point from the rotated system.

Protected Attributes

- GameObject ∗ activeObject

  The active object.
- QPoint lastMousePos

  The last mouse position.
- QPoint offset
- SimulationEngine ∗ simEng

  The simulation engine.
- QGraphicsView ∗ graphView

  The graphics view.
- QStyleOptionGraphicsItem option

  The option for the graphics item.

6.8.1 Detailed Description

A class to represent an overlay widget.

This class provides an interface for creating and managing overlay widgets.

See also

QWidget

Definition at line 27 of file overlaywidget.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 OverlayWidget()

```
OverlayWidget::OverlayWidget (
            QWidget * parent = nullptr,
            SimulationEngine * simEng = nullptr,
            QGraphicsView * graphView = nullptr )  [explicit]
```

Construct a new Overlay Widget object.

Parameters

parent	The parent widget. Default is nullptr.
simEng	The simulation engine. Default is nullptr.
graphView	The graphics view. Default is nullptr.

6.8.3 Member Function Documentation

6.8.3.1 anchor()

```
void OverlayWidget::anchor ( )
```

Anchor the object based on the mouse position back to scene.

Returns

void

6.8.3.2 convertFromRotatedSystem()

```
QPoint OverlayWidget::convertFromRotatedSystem (
             QPoint point,
             qreal angle )  [protected]
```

Convert the point from the rotated system.

Parameters

point	The point in the rotated system.
angle	The angle of the rotation.

Returns

QPoint The point in the rotated system.

6.8.3.3 convertToRotatedSystem()

```
QPoint OverlayWidget::convertToRotatedSystem (
             QPoint point,
             qreal angle )  [protected]
```

Convert the point to the rotated system.

Parameters

point	The point in the scene.
angle	The angle of the rotation.

Returns

QPoint The point in the rotated system.

6.8.3.4 getTimeConstant()

```
qreal * OverlayWidget::getTimeConstant ( )  [inline]
```

Get the time constant of the simulation engine.

Returns

qreal∗ The time constant of the simulation engine.

Definition at line 62 of file overlaywidget.hpp.

```
00062 { return simEng->getTimeConstant(); }
```

6.8.3.5 navigateTheSea()

```
void OverlayWidget::navigateTheSea (
            QMouseEvent * event )
```

Drag the object based on the mouse position in the overlay.

Parameters

event	The mouse event.

Returns

void

6.8.3.6 paintEvent()

```
void OverlayWidget::paintEvent (
            QPaintEvent * event )  [override], [protected]
```

Override the mousePressEvent method.

Parameters

event	The mouse event.

Returns

void

6.8.3.7 setActiveObject()

```
void OverlayWidget::setActiveObject (
             GameObject * obj )  [inline]
```

Get the active object.

Returns

GameObject* The active object.

Definition at line 68 of file overlaywidget.hpp.

```
00068 { activeObject = obj; }
```

6.8.3.8 setLastMousePos()

```
void OverlayWidget::setLastMousePos (
             QPoint pos )  [inline]
```

Get the last mouse position.

Returns

QPoint The last mouse position.

Definition at line 74 of file overlaywidget.hpp.

```
00074 { lastMousePos = pos; }
```

6.8.3.9 trySetSail()

```
void OverlayWidget::trySetSail (
             QMouseEvent * event )
```

Try grab the object based on the mouse position.

Parameters

event	The mouse event.

Returns

void

6.8.4 Member Data Documentation

6.8.4.1 activeObject

```
GameObject* OverlayWidget::activeObject  [protected]
```

The active object.

Definition at line 78 of file overlaywidget.hpp.

6.8.4.2 graphView

```
QGraphicsView* OverlayWidget::graphView  [protected]
```

The graphics view.

Definition at line 89 of file overlaywidget.hpp.

6.8.4.3 lastMousePos

```
QPoint OverlayWidget::lastMousePos  [protected]
```

The last mouse position.

Definition at line 81 of file overlaywidget.hpp.

6.8.4.4 offset

```
QPoint OverlayWidget::offset  [protected]
```

Definition at line 83 of file overlaywidget.hpp.

6.8.4.5 option

```
QStyleOptionGraphicsItem OverlayWidget::option  [protected]
```

The option for the graphics item.

Definition at line 92 of file overlaywidget.hpp.

6.8.4.6 simEng

```
SimulationEngine* OverlayWidget::simEng  [protected]
```

The simulation engine.

Definition at line 86 of file overlaywidget.hpp.

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

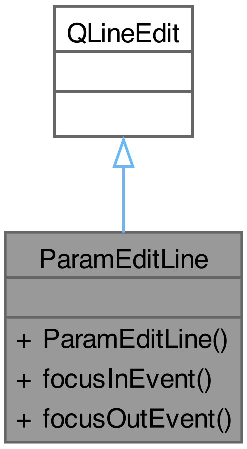
- overlaywidget.hpp

6.9 ParamEditLine Class Reference

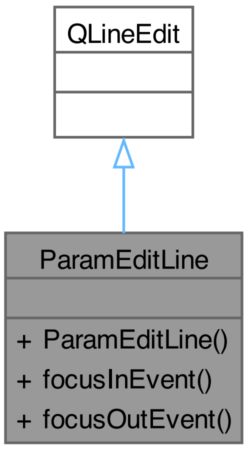
A class to represent a line edit widget for editing parameters.

```
#include <parameditline.hpp>
```

Inheritance diagram for ParamEditLine:



Collaboration diagram for ParamEditLine:



Signals

- void focusIn ()

  Signal emitted when the line edit widget gains focus.
- void focusOut ()

  Signal emitted when the line edit widget loses focus.

Public Member Functions

- ParamEditLine (QWidget ∗parent=nullptr)

  Default constructor.
- void focusInEvent (QFocusEvent ∗event) override

  Overridden focusInEvent method.
- void focusOutEvent (QFocusEvent ∗event) override

  Overridden focusOutEvent method.

6.9.1 Detailed Description

A class to represent a line edit widget for editing parameters.

This class inherits from QLineEdit and provides a line edit widget for editing parameters.

See also

QLineEdit

Definition at line 21 of file parameditline.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 ParamEditLine()

```
ParamEditLine::ParamEditLine (
            QWidget * parent = nullptr )  [inline], [explicit]
```

Default constructor.

Parameters

parent	The parent widget.

Definition at line 29 of file parameditline.hpp.

```
00030         : QLineEdit(parent) {}
```

6.9.3 Member Function Documentation

6.9.3.1 focusIn

```
void ParamEditLine::focusIn ( )  [signal]
```

Signal emitted when the line edit widget gains focus.

Returns

void

6.9.3.2 focusInEvent()

```
void ParamEditLine::focusInEvent (
            QFocusEvent * event )  [inline], [override]
```

Overridden focusInEvent method.

Parameters

event	The focus event.

Returns

void

Definition at line 37 of file parameditline.hpp.

```
                                              {
        QLineEdit::focusOutEvent(event);
        emit focusIn();
    }
```

6.9.3.3 focusOut

```
void ParamEditLine::focusOut ( )  [signal]
```

Signal emitted when the line edit widget loses focus.

Returns

void

6.9.3.4 focusOutEvent()

```
void ParamEditLine::focusOutEvent (
            QFocusEvent * event )  [inline], [override]
```

Overridden focusOutEvent method.

Parameters

event	The focus event.

Returns

void

Definition at line 47 of file parameditline.hpp.

```
                                                {
        QLineEdit::focusOutEvent(event);
        emit focusOut();
    }
```

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

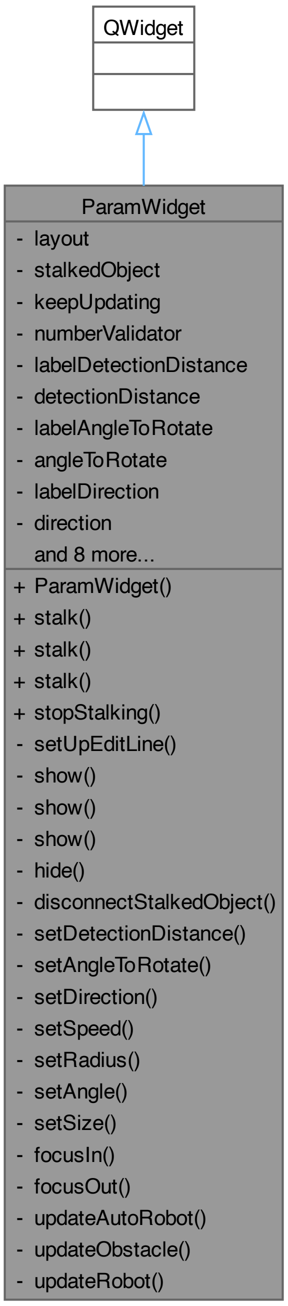
- parameditline.hpp

6.10 ParamWidget Class Reference

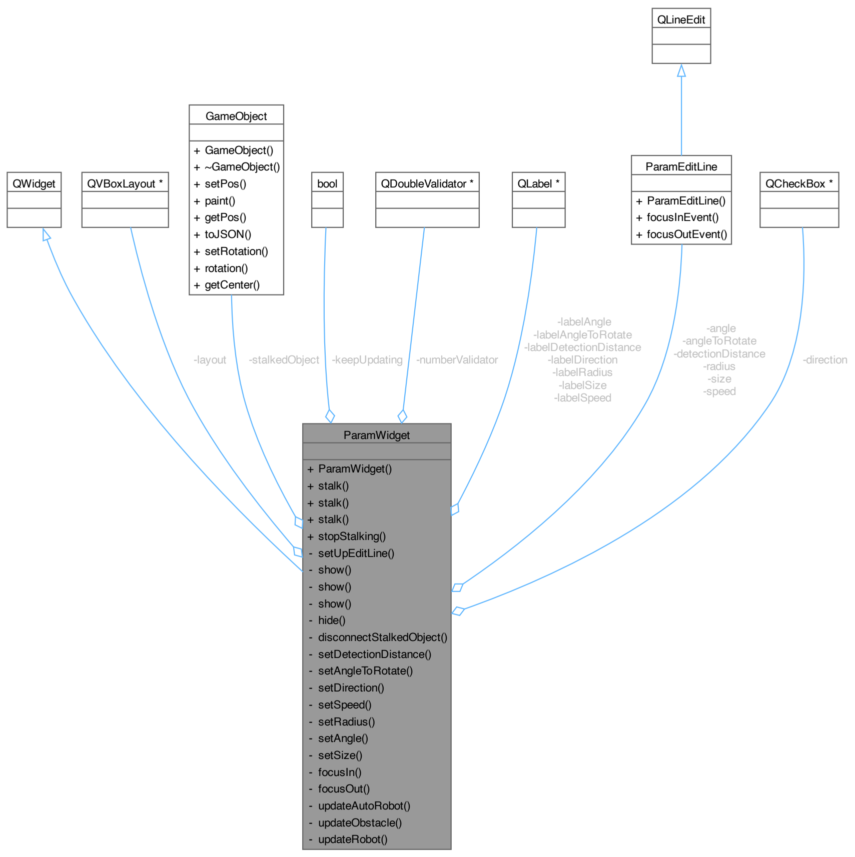
A class to represent a widget for editing parameters of game objects.

```
#include <paramwidget.hpp>
```

Inheritance diagram for ParamWidget:



Collaboration diagram for ParamWidget:



Public Member Functions

- ParamWidget (QWidget ∗parent=nullptr)

  Default constructor.
- void stalk (AutoRobot ∗robot)

  Set the game object whose parameters will be displayed.
- void stalk (Obstacle ∗obstacle)

  Set the game object whose parameters will be displayed.
- void stalk (Robot ∗robot)

  Set the game object whose parameters will be displayed.
- void stopStalking ()

  Stop editing the parameters of the game object.

Private Slots

- void setDetectionDistance ()

  Signal to set the detection distance of the game object.
- void setAngleToRotate ()

  Signal to set the angle to rotate of the game object.
- void setDirection ()

  Signal to set the direction of the game object.
- void setSpeed ()

  Signal to set the speed of the game object.
- void setRadius ()

  Signal to set the radius of the game object.
- void setAngle ()

  Signal to set the angle of the game object.
- void setSize ()

  Signal to set the size of the game object.
- void focusIn ()

  Signal to update the parameters of the game object.
- void focusOut ()

  Signal to update the parameters of the game object.
- void updateAutoRobot ()

  Update the parameters of the game object.
- void updateObstacle ()

  Update the parameters of the game object.
- void updateRobot ()

  Update the parameters of the game object.

Private Member Functions

- void setUpEditLine (ParamEditLine ∗lineEdit, QLabel ∗label)

  Set up the line edit widget for editing a parameter.
- void show (Robot ∗robot)

  Show the parameters of the game object.
- void show (AutoRobot ∗robot)

  Show the parameters of the game object.
- void show (Obstacle ∗obstacle)

  Show the parameters of the game object.
- void hide ()

  Hide the widget.
- void disconnectStalkedObject ()

  Disconnect the widget from the game object.

Private Attributes

- QVBoxLayout ∗ layout

  The layout of the widget.
- GameObject ∗ stalkedObject = nullptr

  The game object whose parameters are being displayed.
- bool keepUpdating = true

  Whether the widget should keep updating the parameters of the game object.
- QDoubleValidator ∗ numberValidator

  The validator for the number input.
- QLabel ∗ labelDetectionDistance

  The labels and line edit widgets for editing the parameters.
- ParamEditLine ∗ detectionDistance
- QLabel ∗ labelAngleToRotate
- ParamEditLine ∗ angleToRotate

- QLabel ∗ labelDirection
- QCheckBox ∗ direction
- QLabel ∗ labelSpeed
- ParamEditLine ∗ speed
- QLabel ∗ labelRadius
- ParamEditLine ∗ radius
- QLabel ∗ labelAngle
- ParamEditLine ∗ angle
- QLabel ∗ labelSize
- ParamEditLine ∗ size

6.10.1 Detailed Description

A class to represent a widget for editing parameters of game objects.

This class inherits from QWidget and provides a widget for editing parameters of game objects.

See also

QWidget

Definition at line 30 of file paramwidget.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ParamWidget()

```
ParamWidget::ParamWidget (
            QWidget * parent = nullptr )  [explicit]
```

Default constructor.

Parameters

parent	The parent widget.

6.10.3 Member Function Documentation

6.10.3.1 disconnectStalkedObject()

```
void ParamWidget::disconnectStalkedObject ( )  [private]
```

Disconnect the widget from the game object.

Returns

void

6.10.3.2 focusIn

```
void ParamWidget::focusIn ( )  [inline], [private], [slot]
```

Signal to update the parameters of the game object.

Returns

void

Definition at line 184 of file paramwidget.hpp.

```
00184 { keepUpdating = false; }
```

6.10.3.3 focusOut

```
void ParamWidget::focusOut ( )  [inline], [private], [slot]
```

Signal to update the parameters of the game object.

Returns

void

Definition at line 190 of file paramwidget.hpp.

```
00190 { keepUpdating = true; }
```

6.10.3.4 hide()

```
void ParamWidget::hide ( )  [private]
```

Hide the widget.

Returns

void

6.10.3.5 setAngle

```
void ParamWidget::setAngle ( )  [private], [slot]
```

Signal to set the angle of the game object.

Returns

void

6.10.3.6 setAngleToRotate

`void ParamWidget::setAngleToRotate ( )  [private], [slot]`

Signal to set the angle to rotate of the game object.

Returns

void

6.10.3.7 setDetectionDistance

`void ParamWidget::setDetectionDistance ( )  [private], [slot]`

Signal to set the detection distance of the game object.

Returns

void

6.10.3.8 setDirection

`void ParamWidget::setDirection ( )  [private], [slot]`

Signal to set the direction of the game object.

Returns

void

6.10.3.9 setRadius

`void ParamWidget::setRadius ( )  [private], [slot]`

Signal to set the radius of the game object.

Returns

void

6.10.3.10 setSize

`void ParamWidget::setSize ( )  [private], [slot]`

Signal to set the size of the game object.

Returns

void

6.10.3.11 setSpeed

```
void ParamWidget::setSpeed ( )  [private], [slot]
```

Signal to set the speed of the game object.

Returns

void

6.10.3.12 setUpEditLine()

```
void ParamWidget::setUpEditLine (
            ParamEditLine * lineEdit,
            QLabel * label )  [private]
```

Set up the line edit widget for editing a parameter.

Parameters

lineEdit	The line edit widget.
label	The label for the line edit widget.

Returns

void

6.10.3.13 show() [1/3]

```
void ParamWidget::show (
            AutoRobot * robot )  [private]
```

Show the parameters of the game object.

Parameters

robot	The robot whose parameters will be displayed.

Returns

void

6.10.3.14 show() [2/3]

```
void ParamWidget::show (
            Obstacle * obstacle )  [private]
```

Show the parameters of the game object.

Parameters

obstacle	The obstacle whose parameters will be displayed.

Returns

void

6.10.3.15 show() [3/3]

```
void ParamWidget::show (
            Robot * robot )  [private]
```

Show the parameters of the game object.

Parameters

robot	The robot whose parameters will be displayed.

Returns

void

6.10.3.16 stalk() [1/3]

```
void ParamWidget::stalk (
            AutoRobot * robot )
```

Set the game object whose parameters will be displayed.

Parameters

object	The game object.

Returns

void

6.10.3.17 stalk() [2/3]

```
void ParamWidget::stalk (
            Obstacle * obstacle )
```

Set the game object whose parameters will be displayed.

Parameters

object	The game object.

Returns

void

6.10.3.18 stalk() [3/3]

```
void ParamWidget::stalk (
            Robot * robot )
```

Set the game object whose parameters will be displayed.

Parameters

object	The game object.

Returns

void

6.10.3.19 stopStalking()

```
void ParamWidget::stopStalking ( )
```

Stop editing the parameters of the game object.

Returns

void

6.10.3.20 updateAutoRobot

```
void ParamWidget::updateAutoRobot ( )  [private], [slot]
```

Update the parameters of the game object.

Returns

void

6.10.3.21 updateObstacle

```
void ParamWidget::updateObstacle ( )  [private], [slot]
```

Update the parameters of the game object.

Returns

void

6.10.3.22 updateRobot

```
void ParamWidget::updateRobot ( )  [private], [slot]
```

Update the parameters of the game object.

Returns

void

6.10.4 Member Data Documentation

6.10.4.1 angle

```
ParamEditLine* ParamWidget::angle  [private]
```

Definition at line 92 of file paramwidget.hpp.

6.10.4.2 angleToRotate

```
ParamEditLine* ParamWidget::angleToRotate  [private]
```

Definition at line 84 of file paramwidget.hpp.

6.10.4.3 detectionDistance

```
ParamEditLine* ParamWidget::detectionDistance  [private]
```

Definition at line 82 of file paramwidget.hpp.

6.10.4.4 direction

```
QCheckBox* ParamWidget::direction  [private]
```

Definition at line 86 of file paramwidget.hpp.

6.10.4.5 keepUpdating

```
bool ParamWidget::keepUpdating = true  [private]
```

Whether the widget should keep updating the parameters of the game object.

Definition at line 75 of file paramwidget.hpp.

6.10.4.6 labelAngle

```
QLabel* ParamWidget::labelAngle  [private]
```

Definition at line 91 of file paramwidget.hpp.

6.10.4.7 labelAngleToRotate

```
QLabel* ParamWidget::labelAngleToRotate  [private]
```

Definition at line 83 of file paramwidget.hpp.

6.10.4.8 labelDetectionDistance

```
QLabel* ParamWidget::labelDetectionDistance  [private]
```

The labels and line edit widgets for editing the parameters.

Definition at line 81 of file paramwidget.hpp.

6.10.4.9 labelDirection

```
QLabel* ParamWidget::labelDirection  [private]
```

Definition at line 85 of file paramwidget.hpp.

6.10.4.10 labelRadius

```
QLabel* ParamWidget::labelRadius  [private]
```

Definition at line 89 of file paramwidget.hpp.

6.10.4.11 labelSize

```
QLabel* ParamWidget::labelSize  [private]
```

Definition at line 93 of file paramwidget.hpp.

6.10.4.12 labelSpeed

```
QLabel* ParamWidget::labelSpeed  [private]
```

Definition at line 87 of file paramwidget.hpp.

6.10.4.13 layout

```
QVBoxLayout* ParamWidget::layout  [private]
```

The layout of the widget.

Definition at line 69 of file paramwidget.hpp.

6.10.4.14 numberValidator

```
QDoubleValidator* ParamWidget::numberValidator  [private]
```

The validator for the number input.

Definition at line 78 of file paramwidget.hpp.

6.10.4.15 radius

```
ParamEditLine* ParamWidget::radius  [private]
```

Definition at line 90 of file paramwidget.hpp.

6.10.4.16 size

```
ParamEditLine* ParamWidget::size  [private]
```

Definition at line 94 of file paramwidget.hpp.

6.10.4.17 speed

```
ParamEditLine* ParamWidget::speed  [private]
```

Definition at line 88 of file paramwidget.hpp.

6.10.4.18 stalkedObject

```
GameObject* ParamWidget::stalkedObject = nullptr  [private]
```

The game object whose parameters are being displayed.

Definition at line 72 of file paramwidget.hpp.

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

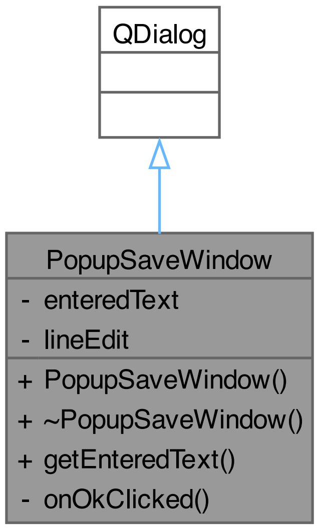
- paramwidget.hpp

6.11 PopupSaveWindow Class Reference

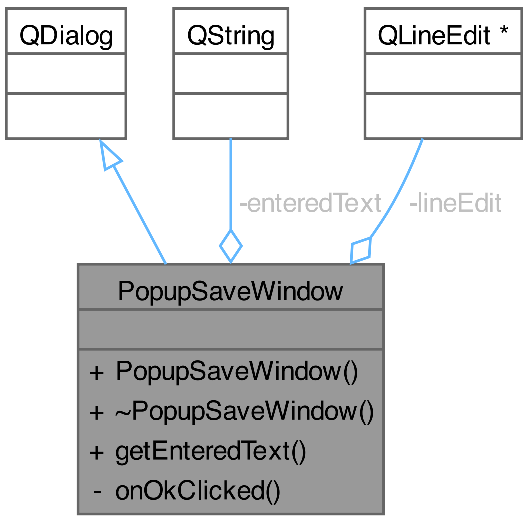
A class to represent a popup save window.

`#include <popupsavewindow.h>`

Inheritance diagram for PopupSaveWindow:



Collaboration diagram for PopupSaveWindow:



Public Member Functions

- PopupSaveWindow (QWidget ∗parent=nullptr)

    Construct a new Popup Save Window object.
- ∼PopupSaveWindow ()
- QString getEnteredText ()

    Get the entered text.

Private Slots

- void onOkClicked ()

    Slot to handle the ok button click event.

Private Attributes

- QString enteredText

    The entered text.
- QLineEdit ∗ lineEdit

    The line edit widget.

6.11.1 Detailed Description

A class to represent a popup save window.

This class provides an interface for creating and managing a popup save window.

See also

QDialog

Definition at line 25 of file popupsavewindow.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 PopupSaveWindow()

```
PopupSaveWindow::PopupSaveWindow (
            QWidget ∗ parent = nullptr )  [explicit]
```

Construct a new Popup Save Window object.

Parameters

parent	The parent widget. Default is nullptr.

6.11.2.2 ∼PopupSaveWindow()

```
PopupSaveWindow::~PopupSaveWindow ( )
```

6.11.3 Member Function Documentation

6.11.3.1 getEnteredText()

```
QString PopupSaveWindow::getEnteredText ( )  [inline]
```

Get the entered text.

Returns

QString The entered text.

Definition at line 40 of file popupsavewindow.h.

```
00040 { return enteredText; }
```

6.11.3.2 onOkClicked

```
void PopupSaveWindow::onOkClicked ( )  [private], [slot]
```

Slot to handle the ok button click event.

Returns

void

6.11.4 Member Data Documentation

6.11.4.1 enteredText

```
QString PopupSaveWindow::enteredText  [private]
```

The entered text.

Definition at line 44 of file popupsavewindow.h.

6.11.4.2 lineEdit

```
QLineEdit* PopupSaveWindow::lineEdit  [private]
```

The line edit widget.

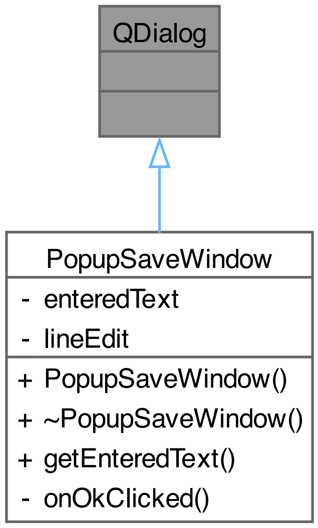
Definition at line 47 of file popupsavewindow.h.

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

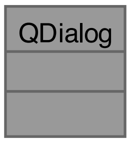
- popupsavewindow.h

6.12 QDialog Class Reference

Inheritance diagram for QDialog:



Collaboration diagram for QDialog:

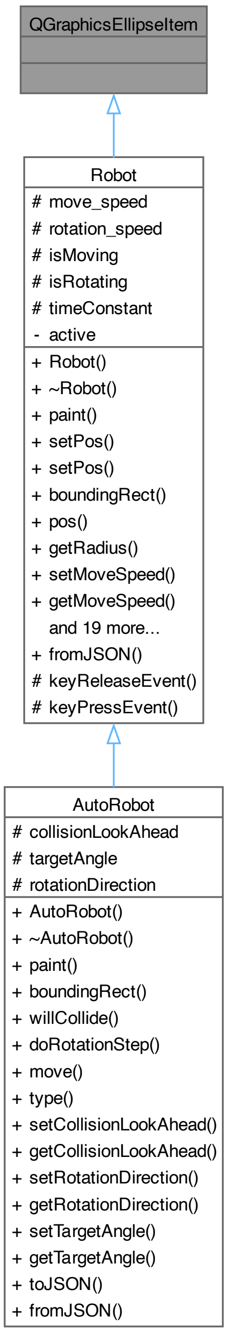


The documentation for this class was generated from the following file:

- popupsavewindow.h

6.13 QGraphicsEllipseItem Class Reference

Inheritance diagram for QGraphicsEllipseItem:



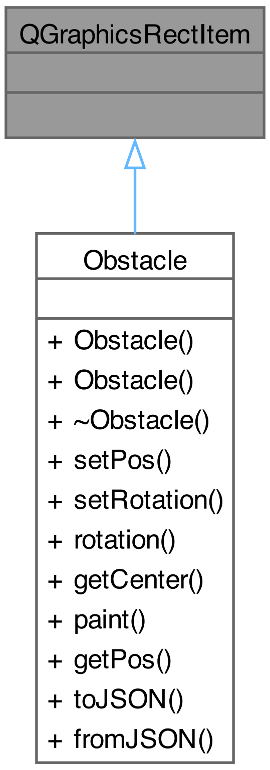
Collaboration diagram for QGraphicsEllipseItem:

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

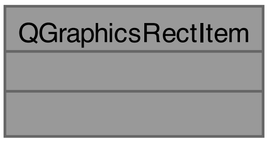
- robot.hpp

6.14 QGraphicsRectItem Class Reference

Inheritance diagram for QGraphicsRectItem:



Collaboration diagram for QGraphicsRectItem:

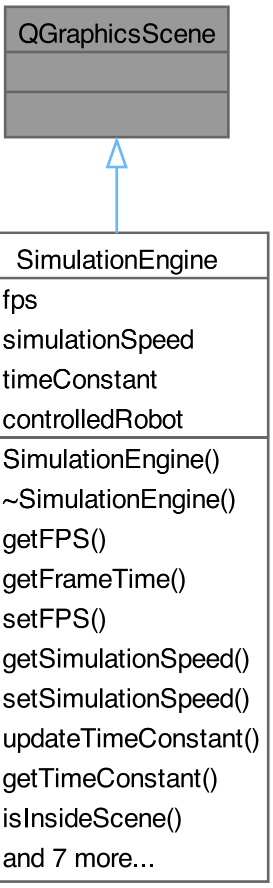


The documentation for this class was generated from the following file:

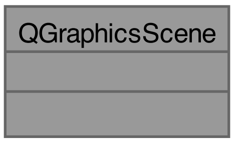
- obstacle.hpp

6.15 QGraphicsScene Class Reference

Inheritance diagram for QGraphicsScene:



Collaboration diagram for QGraphicsScene:

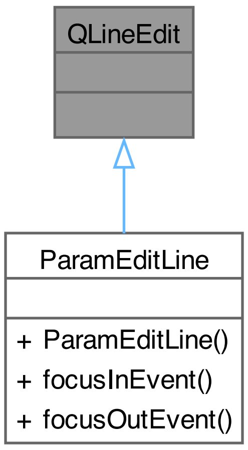


The documentation for this class was generated from the following file:

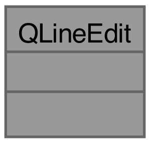
- simulationengine.hpp

6.16 QLineEdit Class Reference

Inheritance diagram for QLineEdit:



Collaboration diagram for QLineEdit:

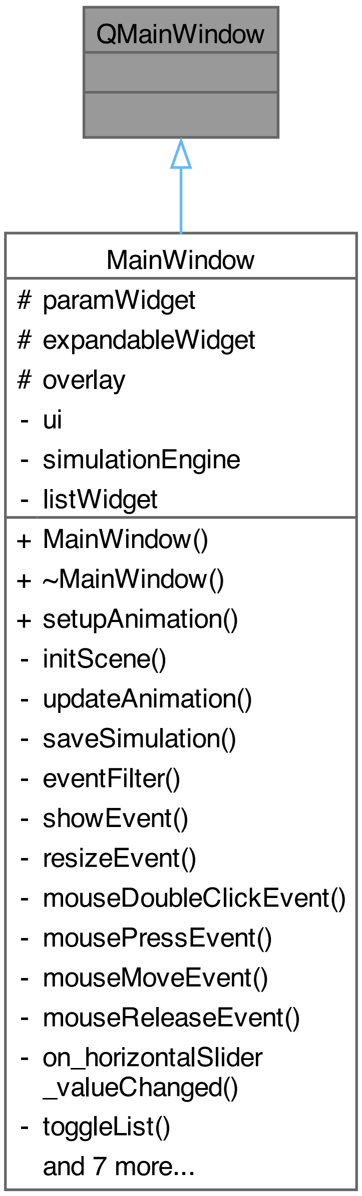


The documentation for this class was generated from the following file:

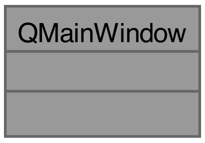
- parameditline.hpp

6.17 QMainWindow Class Reference

Inheritance diagram for QMainWindow:



Collaboration diagram for QMainWindow:

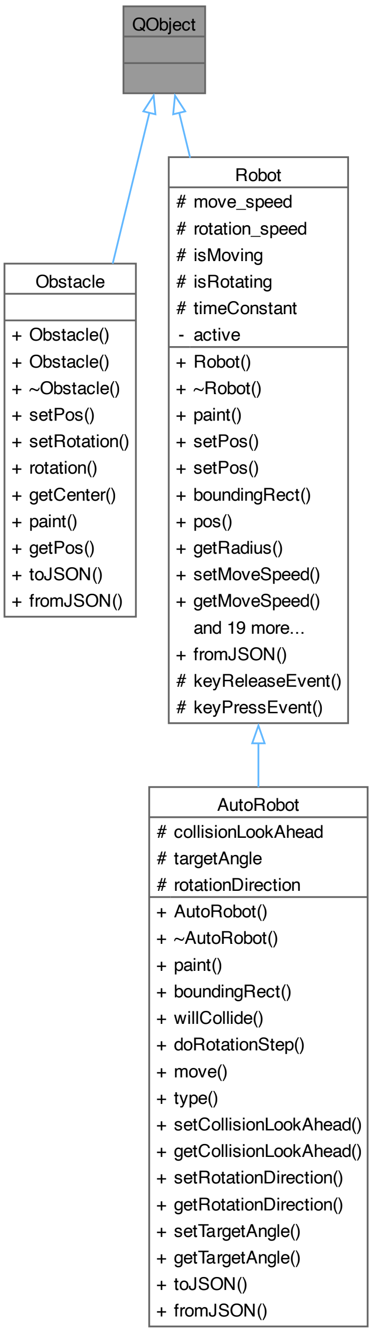


The documentation for this class was generated from the following file:

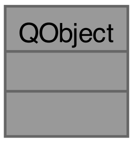
- mainwindow.h

6.18 QObject Class Reference

Inheritance diagram for QObject:



Collaboration diagram for QObject:

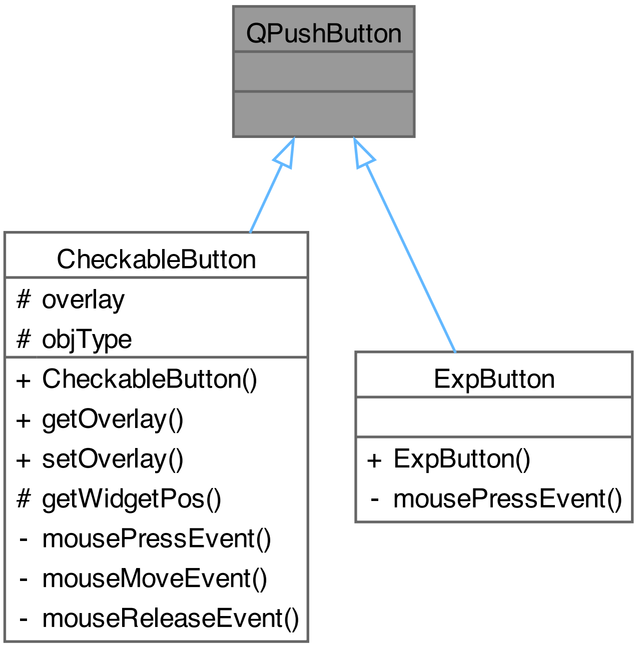


The documentation for this class was generated from the following file:

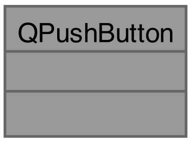
- obstacle.hpp

6.19 QPushButton Class Reference

Inheritance diagram for QPushButton:



Collaboration diagram for QPushButton:

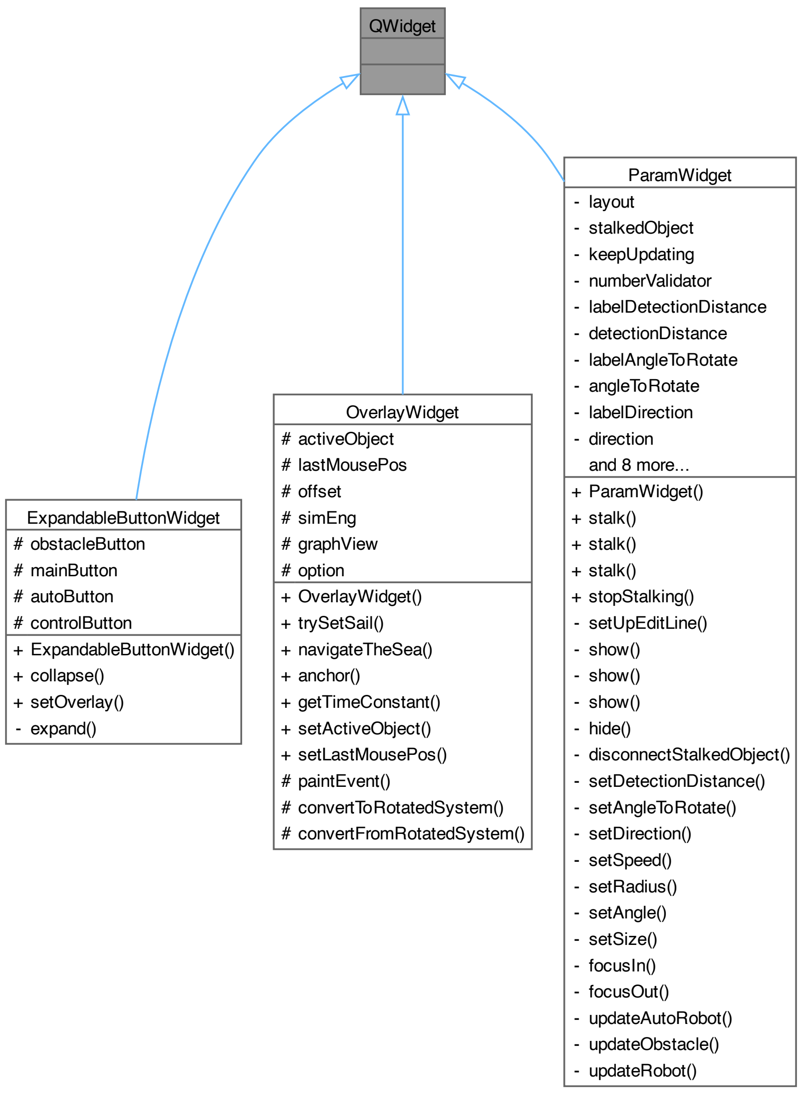


The documentation for this class was generated from the following file:

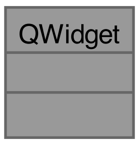
- checkablebutton.hpp

6.20 QWidget Class Reference

Inheritance diagram for QWidget:



Collaboration diagram for QWidget:



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

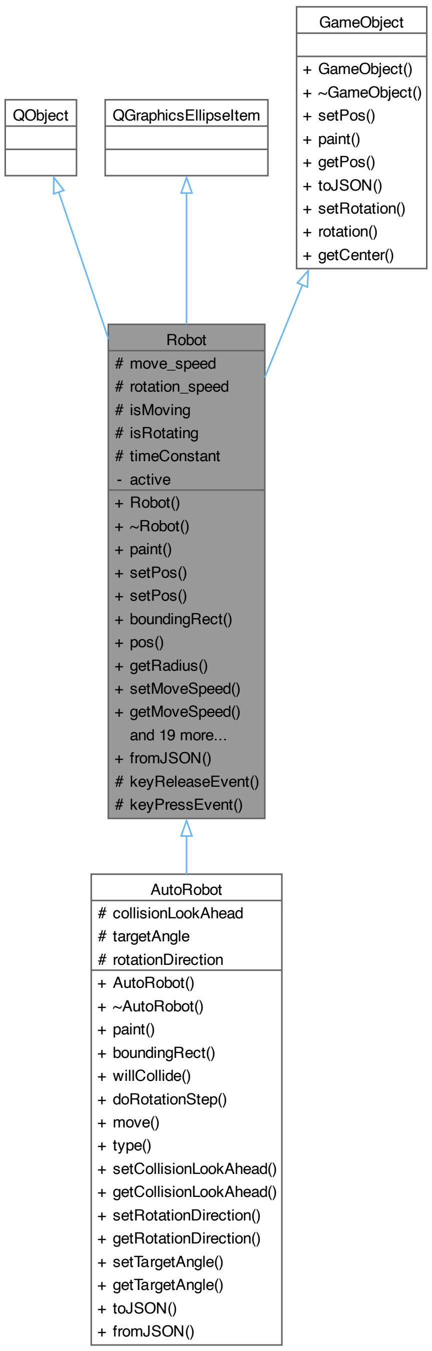
- expbuttonwidget.hpp

6.21 Robot Class Reference

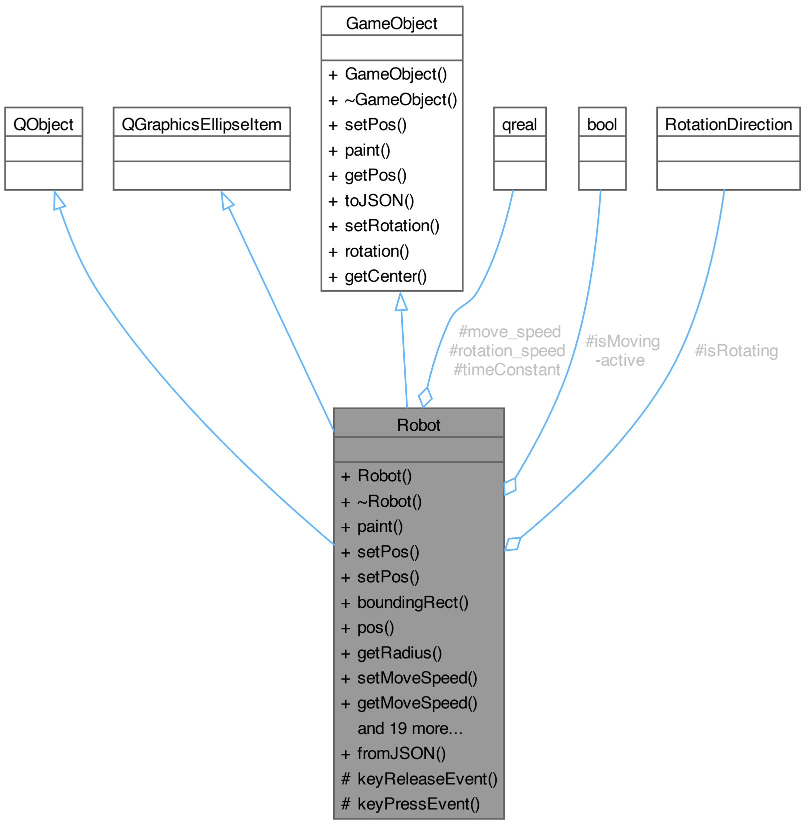
A class to represent a robot in the simulation. By default, the robot is a circle with a line drawn to represent its direction.

```
#include <robot.hpp>
```

Inheritance diagram for Robot:



Collaboration diagram for Robot:



Public Types

- enum RotationDirection { Left = -1 , None = 0 , Right = 1 }

  Enum to represent the direction of rotation of the robot.
- enum { Type = QGraphicsItem::UserType + 1 }

Signals

- void paramsUpdated ()

  Signal emitted when the parameters of the robot are updated.
- void robotSepuku ()

  Signal emitted when the robot is removed.

Public Member Functions

- Robot (QGraphicsItem ∗parent=nullptr, qreal ∗timeConstant=nullptr)

  Default constructor.
- ∼Robot ()
- virtual void paint (QPainter ∗painter, const QStyleOptionGraphicsItem ∗option, QWidget ∗widget) override
- void setPos (const QPointF &pos)

- void setPos (qreal x, qreal y) override
- virtual QRectF boundingRect () const override
- QPointF pos ()
- qreal getRadius () const
- void setMoveSpeed (qreal speed)

  Set the move speed of the robot.
- qreal getMoveSpeed ()

  Get the move speed of the robot.
- void setRotationSpeed (qreal speed)

  Set the rotation speed of the robot.
- qreal getRotationSpeed ()

  Get the rotation speed of the robot.
- void startMoving ()

  Allow the robot to be moved by setting the isMoving flag to true.
- void stopMoving ()

  Stop the robot from moving by setting the isMoving flag to false.
- void startRotating (RotationDirection direction)

  Start rotating the robot in the given direction.
- void stopRotating ()

  Stop the robot from rotating by setting the isRotating flag to None.
- QPointF getDirectionVector ()

  Get the direction vector of the robot.
- virtual bool willCollide (QPointF directionVector, qreal magnitude, bool allowAnticollision=false)

  Check if the robot will collide with any other item in the scene or the scene boundaries if it moves by the given vector.
- virtual bool move ()

  Move the robot based on its current direction and speed. Returns true if the robot moved, false if it didn't (e.g. if it hit a boundary).
- int type () const override

  Get the type of the robot.
- QPointF getPos () override

  Get the position of the robot.
- virtual QJsonObject toJSON () override

  Convert the robot to a JSON object.
- void toggleActive ()

  Toggle the active state of the robot.
- bool isActive ()

  Check if the robot is active.
- qreal getAngle ()

  Get the angle of the robot.
- void setRadius (qreal radius)

  Set the angle of the robot.
- QPointF getCenter () override

  Get the center of the robot.
- qreal rotation () override

  Get the time constant of the simulation.
- void setRotation (qreal angle) override

  Set the rotation of the robot.

Public Member Functions inherited from GameObject

- GameObject ()=default
- ∼GameObject ()=default

Static Public Member Functions

- static Robot ∗ fromJSON (const QJsonObject &object, qreal ∗timeConstant)

  Create a Robot object from a JSON object.

Protected Member Functions

- void keyReleaseEvent (QKeyEvent ∗event)

  The radius of the robot.
- void keyPressEvent (QKeyEvent ∗event)

  Overridden keyPressEvent method.

Protected Attributes

- qreal move_speed = 1

  The speed of the robot.
- qreal rotation_speed = 1

  The speed of the rotation of the robot.
- bool isMoving = false

  Flag to indicate if the robot is moving.
- RotationDirection isRotating = RotationDirection::None

  Flag to indicate the direction of rotation.
- qreal ∗ timeConstant = nullptr

  The time constant of the simulation.

Private Attributes

- bool active = false

  Flag to indicate if the robot is active.

6.21.1 Detailed Description

A class to represent a robot in the simulation. By default, the robot is a circle with a line drawn to represent its direction.

Definition at line 27 of file robot.hpp.

6.21.2 Member Enumeration Documentation

6.21.2.1 anonymous enum

```
anonymous enum
```

Enumerator

Type	

Definition at line 40 of file robot.hpp.

```
00040 { Type = QGraphicsItem::UserType + 1 };
```

6.21.2.2 RotationDirection

```
enum Robot::RotationDirection
```

Enum to represent the direction of rotation of the robot.

Enumerator

Left	
None	
Right	

Definition at line 34 of file robot.hpp.

```
00034                            {
00035         Left = -1, // Counter-clockwise
00036         None = 0,  // No rotation
00037         Right = 1  // Clockwise
00038     };
```

6.21.3 Constructor & Destructor Documentation

6.21.3.1 Robot()

```
Robot::Robot (
            QGraphicsItem * parent = nullptr,
            qreal * timeConstant = nullptr )
```

Default constructor.

Parameters

parent	The parent QGraphicsItem.
timeConstant	The time constant of the simulation.

Returns

void

The time constant is used to calculate the speed of the robot.

6.21.3.2 ∼Robot()

```
Robot::∼Robot ( )
```

6.21.4 Member Function Documentation

6.21.4.1 boundingRect()

```
virtual QRectF Robot::boundingRect ( ) const  [override], [virtual]
```

Reimplemented in AutoRobot.

6.21.4.2 fromJSON()

```
static Robot * Robot::fromJSON (
            const QJsonObject & object,
            qreal * timeConstant )  [static]
```

Create a Robot object from a JSON object.

Parameters

object	The JSON object.
timeConstant	The time constant of the simulation.

Returns

Robot∗

6.21.4.3 getAngle()

```
qreal Robot::getAngle ( )  [inline]
```

Get the angle of the robot.

Returns

qreal

Definition at line 190 of file robot.hpp.

```
00190 { return rotation(); }
```

6.21.4.4 getCenter()

```
QPointF Robot::getCenter ( )  [inline], [override], [virtual]
```

Get the center of the robot.

Returns

QPointF

Implements GameObject.

Definition at line 203 of file robot.hpp.

```
00203 { return boundingRect().center(); }
```

6.21.4.5 getDirectionVector()

`QPointF Robot::getDirectionVector ( )`

Get the direction vector of the robot.

Returns

`QPointF` - Normalized vector representing the direction of the robot on the x and y axes

6.21.4.6 getMoveSpeed()

`qreal Robot::getMoveSpeed ( )`

Get the move speed of the robot.

Returns

qreal

6.21.4.7 getPos()

`QPointF Robot::getPos ( ) [override], [virtual]`

Get the position of the robot.

Returns

QPointF

Implements GameObject.

6.21.4.8 getRadius()

`qreal Robot::getRadius ( ) const`

6.21.4.9 getRotationSpeed()

`qreal Robot::getRotationSpeed ( )`

Get the rotation speed of the robot.

Returns

qreal

6.21.4.10 isActive()

```
bool Robot::isActive ( )  [inline]
```

Check if the robot is active.

Returns

bool

Definition at line 184 of file robot.hpp.

```
00184 { return active; }
```

6.21.4.11 keyPressEvent()

```
void Robot::keyPressEvent (
             QKeyEvent * event )  [protected]
```

Overridden keyPressEvent method.

This method is called when a key is pressed while the robot is focused.

Parameters

event	The key event.

Returns

void

6.21.4.12 keyReleaseEvent()

```
void Robot::keyReleaseEvent (
             QKeyEvent * event )  [protected]
```

The radius of the robot.

6.21.4.13 move()

```
virtual bool Robot::move ( )  [virtual]
```

Move the robot based on its current direction and speed. Returns true if the robot moved, false if it didn't (e.g. if it hit a boundary).

Returns

true

false

Reimplemented in AutoRobot.

6.21.4.14 paint()

```
virtual void Robot::paint (
             QPainter * painter,
             const QStyleOptionGraphicsItem * option,
             QWidget * widget )  [override], [virtual]
```

Override the paint method to draw a line showing the direction of the robot

Implements GameObject.

Reimplemented in AutoRobot.

6.21.4.15 paramsUpdated

```
void Robot::paramsUpdated ( )  [signal]
```

Signal emitted when the parameters of the robot are updated.

Returns

void

6.21.4.16 pos()

```
QPointF Robot::pos ( )
```

Override pos to adjust to center-based positioning

6.21.4.17 robotSepuku

```
void Robot::robotSepuku ( )  [signal]
```

Signal emitted when the robot is removed.

Returns

void

6.21.4.18 rotation()

```
qreal Robot::rotation ( )  [inline], [override], [virtual]
```

Get the time constant of the simulation.

Returns

qreal

Implements GameObject.

Definition at line 209 of file robot.hpp.

```
                                   {
            return QGraphicsEllipseItem::rotation();
        }
```

6.21.4.19 setMoveSpeed()

```
void Robot::setMoveSpeed (
             qreal speed )
```

Set the move speed of the robot.

Parameters

speed	

6.21.4.20 setPos() [1/2]

```
void Robot::setPos (
             const QPointF & pos )
```

Override setPos to adjust to center-based positioning

6.21.4.21 setPos() [2/2]

```
void Robot::setPos (
             qreal x,
             qreal y )  [override], [virtual]
```

Overload setPos to accept x and y coordinates

Implements GameObject.

6.21.4.22 setRadius()

```
void Robot::setRadius (
             qreal radius )
```

Set the angle of the robot.

Parameters

angle	The angle to set.

Returns

void

6.21.4.23 setRotation()

```
void Robot::setRotation (
             qreal angle )  [inline], [override], [virtual]
```

Set the rotation of the robot.

Parameters

angle	The angle to set.

Returns

void

Implements GameObject.

Definition at line 218 of file robot.hpp.

```
00218                                         {
00219         QGraphicsEllipseItem::setRotation(angle);
00220     }
```

6.21.4.24 setRotationSpeed()

```
void Robot::setRotationSpeed (
             qreal speed )
```

Set the rotation speed of the robot.

Parameters

speed	

6.21.4.25 startMoving()

```
void Robot::startMoving ( )
```

Allow the robot to be moved by setting the isMoving flag to true.

6.21.4.26 startRotating()

```
void Robot::startRotating (
             RotationDirection direction )
```

Start rotating the robot in the given direction.

Parameters

direction	

6.21.4.27 stopMoving()

```
void Robot::stopMoving ( )
```

Stop the robot from moving by setting the isMoving flag to false.

6.21.4.28 stopRotating()

```
void Robot::stopRotating ( )
```

Stop the robot from rotating by setting the isRotating flag to None.

6.21.4.29 toggleActive()

```
void Robot::toggleActive ( )  [inline]
```

Toggle the active state of the robot.

If the robot is active, it will be drawn with a light gray fill. If it is inactive, it will be drawn with a transparent fill.

Returns

void

Definition at line 175 of file robot.hpp.

```
                                {
        active = !active;
        active ? setBrush(QBrush(Qt::lightGray)) : setBrush(QBrush(Qt::transparent));
    }
```

6.21.4.30 toJSON()

```
virtual QJsonObject Robot::toJSON ( )  [override], [virtual]
```

Convert the robot to a JSON object.

Returns

QJsonObject

Implements GameObject.

Reimplemented in AutoRobot.

6.21.4.31 type()

```
int Robot::type ( ) const  [inline], [override]
```

Get the type of the robot.

Returns

int

Definition at line 148 of file robot.hpp.

```
{ return Type; }
```

6.21.4.32 willCollide()

```
virtual bool Robot::willCollide (
            QPointF directionVector,
            qreal magnitude,
            bool allowAnticollision = false )  [virtual]
```

Check if the robot will collide with any other item in the scene or the scene boundaries if it moves by the given vector.

Parameters

moveVector	The vector by which the robot will move
allowAnticollision	Flag to indicate if anticollision is allowed

Returns

`true` - if the robot will collide; `false` - if the robot will not collide

Reimplemented in AutoRobot.

6.21.5 Member Data Documentation

6.21.5.1 active

```
bool Robot::active = false  [private]
```

Flag to indicate if the robot is active.

Definition at line 264 of file robot.hpp.

6.21.5.2 isMoving

```
bool Robot::isMoving = false  [protected]
```

Flag to indicate if the robot is moving.

Definition at line 243 of file robot.hpp.

6.21.5.3 isRotating

```
RotationDirection Robot::isRotating = RotationDirection::None  [protected]
```

Flag to indicate the direction of rotation.

Definition at line 246 of file robot.hpp.

6.21.5.4 move_speed

```
qreal Robot::move_speed = 1  [protected]
```

The speed of the robot.

Definition at line 238 of file robot.hpp.

6.21.5.5 rotation_speed

```
qreal Robot::rotation_speed = 1  [protected]
```

The speed of the rotation of the robot.

Definition at line 240 of file robot.hpp.

6.21.5.6 timeConstant

```
qreal* Robot::timeConstant = nullptr  [protected]
```

The time constant of the simulation.

Definition at line 249 of file robot.hpp.

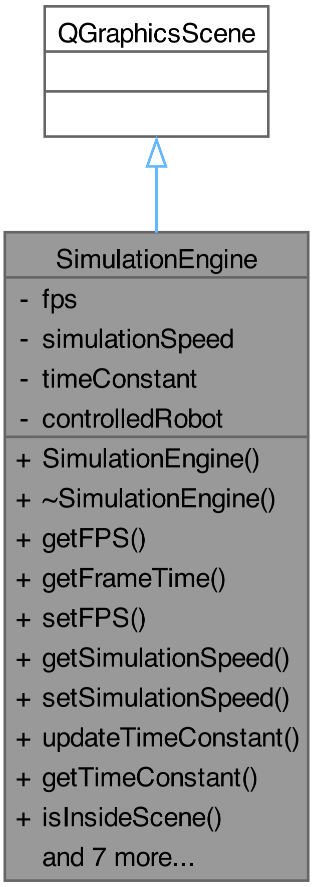
The documentation for this class was generated from the following file:

- robot.hpp

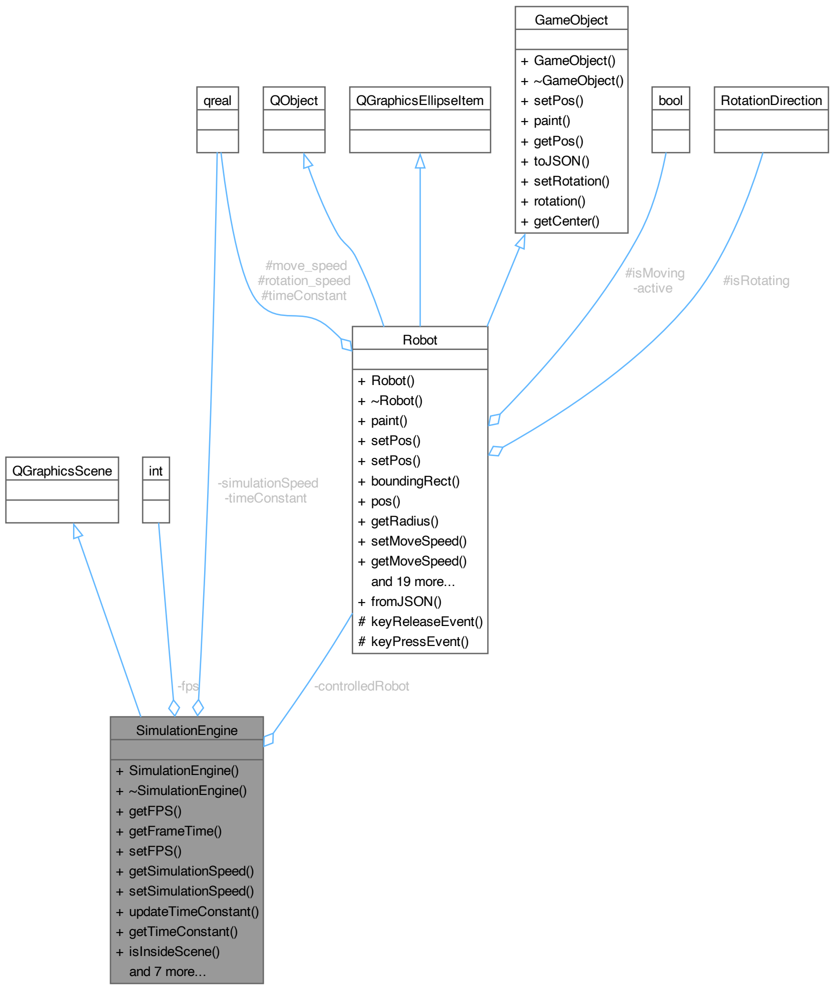
6.22 SimulationEngine Class Reference

```
#include <simulationengine.hpp>
```

Inheritance diagram for SimulationEngine:



Collaboration diagram for SimulationEngine:



Public Member Functions

- SimulationEngine (QObject ∗parent=nullptr, int fps=60, qreal simulationSpeed=1.0/16.0)
- ∼SimulationEngine ()
- int getFPS ()

    Simulation Frames-Per-Second getter.
- int getFrameTime ()

    Get the time it takes to render a single frame.
- void setFPS (int fps)

    Set the simulation Frames-Per-Second.
- qreal getSimulationSpeed ()

    Get the simulation speed.
- void setSimulationSpeed (qreal speed)

    Set the simulation speed.
- void updateTimeConstant ()

    Update the time constant.
- qreal ∗ getTimeConstant ()

Get the time constant pointer.
- bool isInsideScene (const QPointF &point) const

  Check if a point is inside the scene.
- Robot ∗ getControlledRobot ()

  Get the robot that is currently being controlled.
- void setControlledRobot (Robot ∗robot)

  Set the robot that is currently being controlled.
- bool saveSimulation (const QString &filename="simulation")

  Save the simulation.
- bool loadSimulation (QString filename="simulation")

  Load the simulation.
- void read (const QJsonObject &json)

  Read the simulation from a JSON object.
- QJsonObject toJson () const

  Convert the simulation to a JSON object.
- void clearScene ()

  Clear the scene.

Private Attributes

- int fps = 60
- qreal simulationSpeed = 1
- qreal timeConstant = 1
- Robot ∗ controlledRobot = nullptr

6.22.1 Detailed Description

Definition at line 19 of file simulationengine.hpp.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 SimulationEngine()

```
SimulationEngine::SimulationEngine (
            QObject * parent = nullptr,
            int fps = 60,
            qreal simulationSpeed = 1.0/16.0 )
```

6.22.2.2 ∼SimulationEngine()

```
SimulationEngine::∼SimulationEngine ( )
```

6.22.3 Member Function Documentation

6.22.3.1 clearScene()

```
void SimulationEngine::clearScene ( )
```

Clear the scene.

6.22.3.2 getControlledRobot()

```
Robot * SimulationEngine::getControlledRobot ( )
```

Get the robot that is currently being controlled.

Returns

Robot*

6.22.3.3 getFPS()

```
int SimulationEngine::getFPS ( )
```

Simulation Frames-Per-Second getter.

Returns

int

6.22.3.4 getFrameTime()

```
int SimulationEngine::getFrameTime ( )
```

Get the time it takes to render a single frame.

Returns

int

6.22.3.5 getSimulationSpeed()

```
qreal SimulationEngine::getSimulationSpeed ( )
```

Get the simulation speed.

Returns

qreal

6.22.3.6 getTimeConstant()

```
qreal * SimulationEngine::getTimeConstant ( )
```

Get the time constant pointer.

Returns

qreal*

6.22.3.7 isInsideScene()

```
bool SimulationEngine::isInsideScene (
             const QPointF & point ) const
```

Check if a point is inside the scene.

Parameters

point	

Returns

bool

6.22.3.8 loadSimulation()

```
bool SimulationEngine::loadSimulation (
            QString filename = "simulation" )
```

Load the simulation.

Parameters

filename	The name of the file to load the simulation from.

The file will be loaded from the JSON format from folders "simulations" and "exmaples"

Returns

void

6.22.3.9 read()

```
void SimulationEngine::read (
            const QJsonObject & json )
```

Read the simulation from a JSON object.

Parameters

json	The JSON object to read.

Returns

void

6.22.3.10 saveSimulation()

```
bool SimulationEngine::saveSimulation (
            const QString & filename = "simulation" )
```

Save the simulation.

Parameters

filename	The name of the file to save the simulation to.

The file will be saved in the JSON format in folder "simulations"

Returns

void

6.22.3.11 setControlledRobot()

```
void SimulationEngine::setControlledRobot (
             Robot * robot )
```

Set the robot that is currently being controlled.

Parameters

robot	

Returns

void

6.22.3.12 setFPS()

```
void SimulationEngine::setFPS (
             int fps )
```

Set the simulation Frames-Per-Second.

Parameters

fps	

6.22.3.13 setSimulationSpeed()

```
void SimulationEngine::setSimulationSpeed (
             qreal speed )
```

Set the simulation speed.

Parameters

speed	

Returns

void

6.22.3.14 toJson()

```
QJsonObject SimulationEngine::toJson ( ) const
```

Convert the simulation to a JSON object.

Returns

QJsonObject

6.22.3.15 updateTimeConstant()

```
void SimulationEngine::updateTimeConstant ( )
```

Update the time constant.

Returns

void

6.22.4 Member Data Documentation

6.22.4.1 controlledRobot

```
Robot* SimulationEngine::controlledRobot = nullptr  [private]
```

The robot that is currently being controlled.

Definition at line 133 of file simulationengine.hpp.

6.22.4.2 fps

```
int SimulationEngine::fps = 60  [private]
```

The frames per second of the simulation engine.

Definition at line 125 of file simulationengine.hpp.

6.22.4.3 simulationSpeed

```
qreal SimulationEngine::simulationSpeed = 1  [private]
```

The speed of the simulation engine.

Definition at line 127 of file simulationengine.hpp.

6.22.4.4 timeConstant

```
qreal SimulationEngine::timeConstant = 1  [private]
```

The time constant of the simulation engine.

Definition at line 130 of file simulationengine.hpp.

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

- simulationengine.hpp

Chapter 7

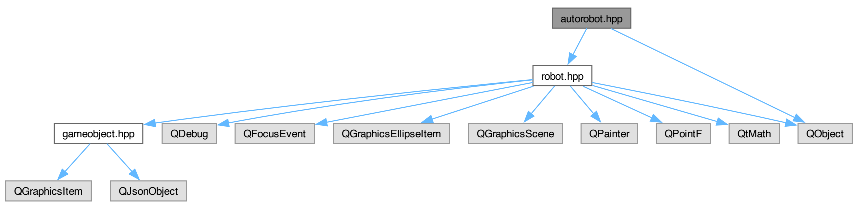
File Documentation

7.1 autorobot.hpp File Reference

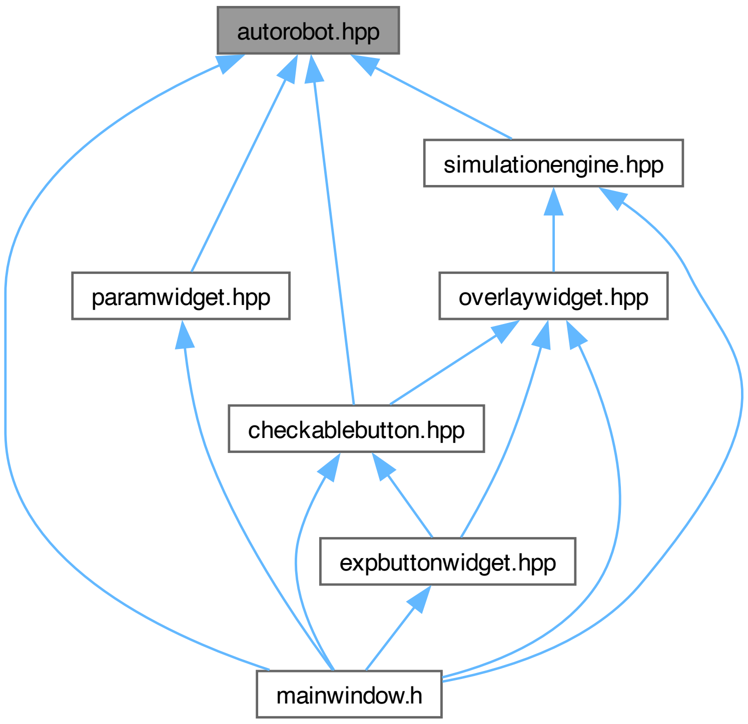
This file contains the declaration of the AutoRobot class.

```
#include "robot.hpp"
#include <QObject>
```

Include dependency graph for autorobot.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class AutoRobot

  A class to represent an autonomous robot.

Macros

- #define SMOOTH_ROTATION_SPEED 0.25

7.1.1 Detailed Description

This file contains the declaration of the AutoRobot class.

It is a subclass of the Robot class and represents an autonomous robot.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file autorobot.hpp.

7.1.2 Macro Definition Documentation

7.1.2.1 SMOOTH_ROTATION_SPEED

```
#define SMOOTH_ROTATION_SPEED 0.25
```

Definition at line 15 of file autorobot.hpp.

7.2 autorobot.hpp

Go to the documentation of this file.

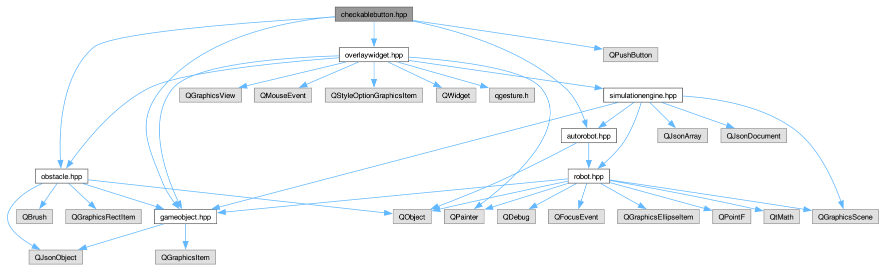
```
00001 /**
00002  * @file autorobot.hpp
00003  * @brief This file contains the declaration of the AutoRobot class.
00004  * @details It is a subclass of the Robot class and represents an autonomous robot.
00005  * @authors Tomáš Hobza, Jakub Všetečka
00006  * @date 02.05.2024
00007  */
00008
00009 #ifndef AUTOROBOT_HPP
00010 #define AUTOROBOT_HPP
00011
00012 #include "robot.hpp"
00013 #include <QObject>
00014
00015 #define SMOOTH_ROTATION_SPEED 0.25
00016
00017 /**
00018  * @class AutoRobot
00019  * @brief A class to represent an autonomous robot.
00020  * @details This class inherits from Robot and provides functionalities for an autonomous robot.
00021  * @see Robot
00022  */
00023 class AutoRobot : public Robot {
00024     // Q_OBJECT
00025
00026   public:
00027     enum { Type = QGraphicsItem::UserType + 2 };
00028
00029     /**
00030      * @brief Constructor for AutoRobot.
00031      * @param parent The parent QGraphicsItem.
00032      * @param size The size of the robot.
00033      * @param collisionLookAhead The distance the robot looks ahead for collisions.
00034      * @param rotationDirection The initial rotation direction of the robot.
00035      * @param moveSpeed The movement speed of the robot.
00036      * @param rotationSpeed The rotation speed of the robot.
00037      * @param timeConstant A pointer to the time constant.
00038      */
00039     AutoRobot(QGraphicsItem *parent = nullptr, qreal size = 50, qreal collisionLookAhead = 10,
      Robot::RotationDirection rotationDirection = Robot::RotationDirection::Right, qreal moveSpeed = 1,
      qreal rotationSpeed = 1, qreal *timeConstant = nullptr);
00040     ~AutoRobot();
00041
00042     /** Override the paint method to draw a line showing the direction of the robot */
00043     void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
00044
00045     /* Override the boundingRect method to adjust the bounding rectangle */
00046     QRectF boundingRect() const override;
00047
00048     /**
00049      * @brief Check if the robot will collide with any object in the scene
00050      * @param directionVector The direction vector of the robot
00051      * @param magnitude The magnitude of the direction vector
00052      * @param allowAnticollision Whether to allow anticollision
00053      * @return bool Whether the robot will collide with any object in the scene
00054      */
00055     bool willCollide(QPointF directionVector, qreal magnitude, bool allowAnticollision) override;
00056
00057     /**
00058      * @brief Perform a rotation step
00059      * @param direction The direction of the rotation
00060      * @return void
00061      */
00062     void doRotationStep(RotationDirection direction);
00063
00064     /**
```

```
     * @brief Perform a movement step
     * @return bool Whether the movement step was successful
     */
    bool move() override;

    /**
     * @brief Get the type of the object
     * @return int The type of the object
     */
    int type() const override { return Type; }

    /**
     * @brief Set the look ahead distance for collision detection
     * @param lookAhead The look ahead distance
     * @return void
     */
    void setCollisionLookAhead(qreal lookAhead) { collisionLookAhead = lookAhead; }

    /**
     * @brief Get the look ahead distance for collision detection
     * @return qreal The look ahead distance
     */
    qreal getCollisionLookAhead() { return collisionLookAhead; }

    /**
     * @brief Set the rotation direction of the robot
     * @param direction The rotation direction
     * @return void
     */
    void setRotationDirection(RotationDirection direction) { rotationDirection = direction; }

    /**
     * @brief Get the rotation direction of the robot
     * @return RotationDirection The rotation direction
     */
    RotationDirection getRotationDirection() { return rotationDirection; }

    /**
     * @brief Set the target angle of the robot
     * @param angle The target angle
     * @return void
     */
    void setTargetAngle(qreal angle) { targetAngle = angle; }

    /**
     * @brief Get the target angle of the robot
     * @return qreal The target angle
     */
    qreal getTargetAngle() { return targetAngle; }

    /**
     * @brief Get the JSON representation of the object
     * @return QJsonObject The JSON representation of the object
     */
    QJsonObject toJSON() override;

    /**
     * @brief Create an AutoRobot object from a JSON object
     * @param object The JSON object to create the AutoRobot object from
     * @param timeConstant The time constant of the robot
     * @return AutoRobot* The AutoRobot object created from the JSON object
     */
    static AutoRobot *fromJSON(const QJsonObject &object, qreal *timeConstant);

  protected:
    /** @brief The look ahead distance for collision detection */
    qreal collisionLookAhead = 0;

    /** @brief The target angle of the robot */
    qreal targetAngle = 0;

    /** @brief The rotation direction of the robot */
    Robot::RotationDirection rotationDirection = Robot::RotationDirection::Right;
};

#endif // AUTOROBOT_HPP
```

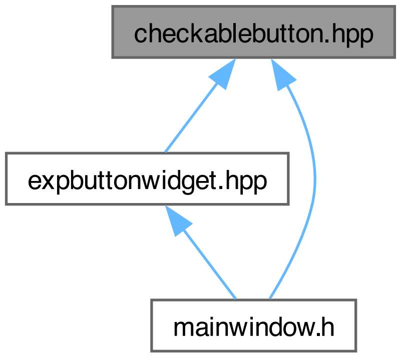
7.3 checkablebutton.hpp File Reference

This file contains the declaration of the CheckableButton class.

```
#include "autorobot.hpp"
#include "gameobject.hpp"
#include "obstacle.hpp"
#include "overlaywidget.hpp"
#include <QPushButton>
```
Include dependency graph for checkablebutton.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CheckableButton

  A class to represent a checkable button.

7.3.1 Detailed Description

This file contains the declaration of the CheckableButton class.

It is a subclass of the QPushButton class and represents a checkable button.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file checkablebutton.hpp.

7.4 checkablebutton.hpp

Go to the documentation of this file.

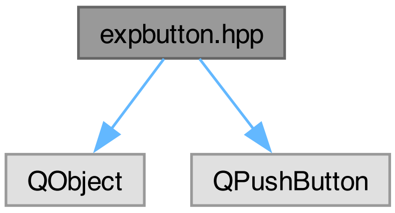
```
/**
 * @file checkablebutton.hpp
 * @brief This file contains the declaration of the CheckableButton class.
 * @details It is a subclass of the QPushButton class and represents a checkable button.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef CHECKABLEBUTTON_HPP
#define CHECKABLEBUTTON_HPP

#include "autorobot.hpp"
#include "gameobject.hpp"
#include "obstacle.hpp"
#include "overlaywidget.hpp"
#include <QPushButton>

/**
 * @class CheckableButton
 * @brief A class to represent a checkable button.
 * @details This class inherits from QPushButton and provides functionalities for a button that can be
checked and unchecked. It also has an OverlayWidget that is used to draw the object on the grid.
 * @see QPushButton
 */
class CheckableButton : public QPushButton {
  public:
    /** * @brief Enum to represent the type of object that the button represents
     * AUTO: AutoRobot
     * CONT: ControlledRobot
     * OBST: Obstacle
     */
    enum ObjectType {
        AUTO,
        CONT,
        OBST
    };

    /**
     * @brief Constructor for CheckableButton.
     * @param text The text to be displayed on the button.
     * @param parent The parent QWidget.
     * @param type The type of object that the button represents.
     */
    explicit CheckableButton(const QString &text, QWidget *parent = nullptr, ObjectType type =
ObjectType::OBST);

    /**
     * @brief Get the overlay widget of the button
     * @return OverlayWidget* The overlay widget of the button
     */
    OverlayWidget *getOverlay() const { return overlay; }

    /**
     * @brief Set the overlay widget of the button
     * @param overlay The overlay widget to set
     * @return void
     */
    void setOverlay(OverlayWidget *overlay) { this->overlay = overlay; }

  protected:
    /** @brief Pointer to the overlay widget */
    OverlayWidget *overlay;

    /** @brief The type of object that the button represents */
    ObjectType objType;

    /**
     * @brief Get the position of the widget on the grid.
     * @param localPos The local position of the mouse.
     * @return QPoint The position in the overlay widget.
     */
    QPoint getWidgetPos(QPoint localPos);

  private slots:

    /**
     * @brief Override the mousePressEvent method
     * @param event The mouse event
     * @return void
     */
    void mousePressEvent(QMouseEvent *event) override;

    /**
     * @brief Override the mouseMoveEvent method
     * @param event The mouse event
     * @return void
     */
    void mouseMoveEvent(QMouseEvent *event) override;

    /**
     * @brief Override the mouseReleaseEvent method
     * @param event The mouse event
     * @return void
     */
    void mouseReleaseEvent(QMouseEvent *event) override;
};

#endif // CHECKABLEBUTTON_HPP
```

7.5 expbutton.hpp File Reference

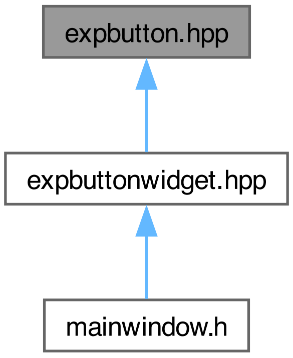
This file contains the declaration of the ExpButton class.

```
#include <QObject>
#include <QPushButton>
```

Include dependency graph for expbutton.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ExpButton

  A class for expandable buttons.

7.5.1 Detailed Description

This file contains the declaration of the ExpButton class.

It is a subclass of the QPushButton class and represents an expandable button.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file expbutton.hpp.

7.6 expbutton.hpp

Go to the documentation of this file.

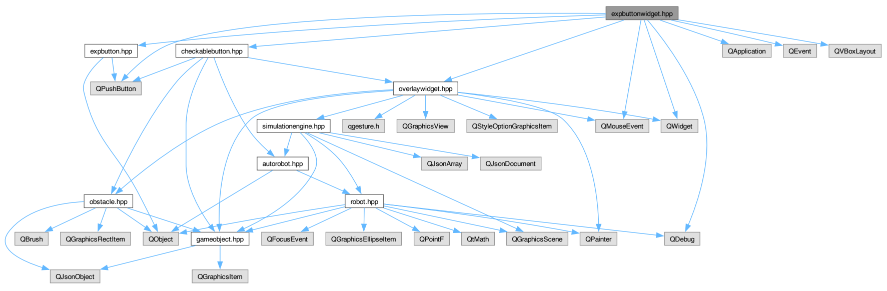
```
/**
 * @file expbutton.hpp
 * @brief This file contains the declaration of the ExpButton class.
 * @details It is a subclass of the QPushButton class and represents an expandable button.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef EXPBUTTON_HPP
#define EXPBUTTON_HPP

#include <QObject>
#include <QPushButton>

/**
 * @class ExpButton
 * @brief A class for expandable buttons.
 * @details This class inherits from QPushButton and emits a signal when pressed.
 * @see QPushButton
 */
class ExpButton : public QPushButton {
    Q_OBJECT

  public:
    /**
     * @brief Constructor for ExpButton.
     * @param text The text to be displayed on the button.
     * @param parent The parent QWidget.
     */
    explicit ExpButton(const QString &text, QWidget *parent = nullptr);

  signals:
    /**
     * @brief Signal emitted when the button is pressed.
     * @return void
     */
    void pressed();

  private slots:
    /**
     * @brief Slot to handle the button press event.
     * @param event The QMouseEvent that triggered the slot.
     * @return void
     */
    void mousePressEvent(QMouseEvent *event) override;
};

#endif // EXPBUTTON_HPP
```

7.7 expbuttonwidget.hpp File Reference

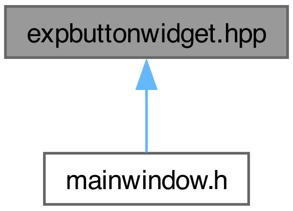
This file contains the declaration of the ExpandableButtonWidget class.

```
#include "checkablebutton.hpp"
#include "expbutton.hpp"
#include "overlaywidget.hpp"
#include <QApplication>
#include <QDebug>
#include <QEvent>
#include <QMouseEvent>
#include <QPushButton>
#include <QVBoxLayout>
#include <QWidget>
```

Include dependency graph for expbuttonwidget.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ExpandableButtonWidget

  A class to represent an expandable button widget.

7.7.1 Detailed Description

This file contains the declaration of the ExpandableButtonWidget class.

It is a subclass of the QWidget class and represents an expandable button widget.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file expbuttonwidget.hpp.

7.8 expbuttonwidget.hpp

Go to the documentation of this file.

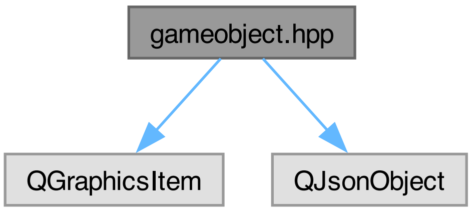
```
/**
 * @file expbuttonwidget.hpp
 * @brief This file contains the declaration of the ExpandableButtonWidget class.
 * @details It is a subclass of the QWidget class and represents an expandable button widget.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef EXPANDABLEBUTTONWIDGET_HPP
#define EXPANDABLEBUTTONWIDGET_HPP

// ExpandableButtonWidget.h
#include "checkablebutton.hpp"
#include "expbutton.hpp"
#include "overlaywidget.hpp"
#include <QApplication>
#include <QDebug>
#include <QEvent>
#include <QMouseEvent>
#include <QPushButton>
#include <QVBoxLayout>
#include <QWidget>

/**
 * @class ExpandableButtonWidget
 * @brief A class to represent an expandable button widget.
 * @details This class provides an interface for creating and managing expandable button widgets.
 * @see QWidget
 */
class ExpandableButtonWidget : public QWidget {
    Q_OBJECT

  public:
    /**
     * @brief Construct a new Expandable Button Widget object.
     * @param parent The parent widget. Default is nullptr.
     */
    explicit ExpandableButtonWidget(QWidget *parent = nullptr);

    /**
     * @brief Get the obstacle button.
     * @return CheckableButton* The obstacle button.
     */
    void collapse();

    /**
     * @brief Get the obstacle button.
     * @return CheckableButton* The obstacle button.
     */
    void setOverlay(OverlayWidget *overlay);

  protected:
    /** @brief Reference to the obstacle button.*/
    CheckableButton *obstacleButton;

    /** @brief Reference to the main button.*/
    ExpButton *mainButton;

    /** @brief Reference to the auto button.*/
    CheckableButton *autoButton;

    /** @brief Reference to the control button.*/
    CheckableButton *controlButton;

  private slots:
    /**
     * @brief Slot to handle the main button press event.
     * @return void
     */
    void expand();
};

#endif // EXPANDABLEBUTTONWIDGET_HPP
```

7.9 gameobject.hpp File Reference

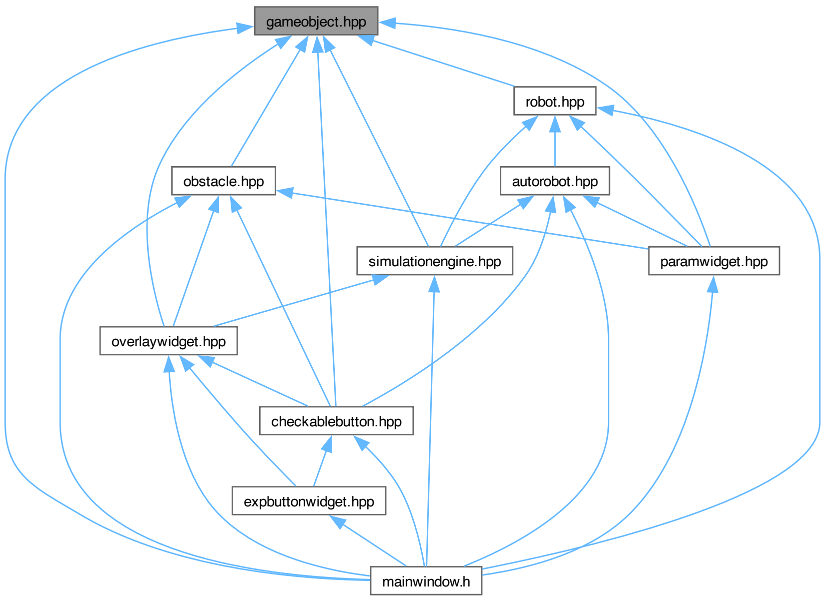
This file contains the declaration of the GameObject class.

```
#include <QGraphicsItem>
#include <QJsonObject>
```

Include dependency graph for gameobject.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class GameObject

  A class to represent a game object in the simulation.

7.9.1 Detailed Description

This file contains the declaration of the GameObject class.

It is an abstract class that represents a game object in the simulation.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file gameobject.hpp.

7.10 gameobject.hpp

Go to the documentation of this file.

```
/**
 * @file gameobject.hpp
 * @brief This file contains the declaration of the GameObject class.
 * @details It is an abstract class that represents a game object in the simulation.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef GAMEOBJECT_HPP
#define GAMEOBJECT_HPP

#include <QGraphicsItem>
#include <QJsonObject>

/**
 * @class GameObject
 * @brief A class to represent a game object in the simulation.
 * @details This class provides an interface for creating and managing game objects.
 */
class GameObject {

  public:
    GameObject() = default;
    ~GameObject() = default;

    /**
     * @brief Set the position of the game object.
     * @param x
     * @param y
     * @return void
     */
    virtual void setPos(qreal x, qreal y) = 0;

    /**
     * @brief Paint the game object.
     * @param painter
     * @param option
     * @param widget
     * @return void
     */
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) =
0;

    /**
     * @brief Get the position of the game object.
     * @return QPointF
     */
    virtual QPointF getPos() = 0;

    /**
     * @brief Convert the game object to a JSON object.
     * @return QJsonObject
     */
    virtual QJsonObject toJSON() = 0;

    /**
     * @brief Set the rotation of the game object.
     * @param angle
     * @return void
     */
    virtual void setRotation(qreal angle) = 0;

    /**
```

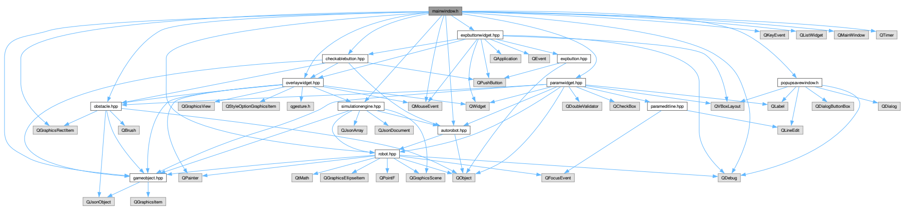
```
     * @brief Get the rotation of the game object.
     * @return qreal
     */
    virtual qreal rotation() = 0;

    /**
     * @brief Get the center of the game object.
     * @return QPointF
     */
    virtual QPointF getCenter() = 0;
};

#endif // GAMEOBJECT_HPP
```

7.11 mainwindow.h File Reference

This file contains the declaration of the MainWindow class.

```
#include "autorobot.hpp"
#include "checkablebutton.hpp"
#include "expbuttonwidget.hpp"
#include "gameobject.hpp"
#include "obstacle.hpp"
#include "overlaywidget.hpp"
#include "paramwidget.hpp"
#include "popupsavewindow.h"
#include "robot.hpp"
#include "simulationengine.hpp"
#include <QDebug>
#include <QGraphicsRectItem>
#include <QKeyEvent>
#include <QListWidget>
#include <QMainWindow>
#include <QMouseEvent>
#include <QTimer>
```

Include dependency graph for mainwindow.h:



Classes

- class MainWindow

  A class to represent the main window of the application.

Namespaces

- namespace Ui

7.11.1 Detailed Description

This file contains the declaration of the MainWindow class.

It is a subclass of the QMainWindow class and represents the main window of the application.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file mainwindow.h.

7.12 mainwindow.h

Go to the documentation of this file.

```
/**
 * @file mainwindow.h
 * @brief This file contains the declaration of the MainWindow class.
 * @details It is a subclass of the QMainWindow class and represents the main window of the
application.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "autorobot.hpp"
#include "checkablebutton.hpp"
#include "expbuttonwidget.hpp"
#include "gameobject.hpp"
#include "obstacle.hpp"
#include "overlaywidget.hpp"
#include "paramwidget.hpp"
#include "popupsavewindow.h"
#include "robot.hpp"
#include "simulationengine.hpp"
#include <QDebug>
#include <QGraphicsRectItem>

#include <QKeyEvent>
#include <QListWidget>
#include <QMainWindow>
#include <QMouseEvent>
#include <QTimer>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

/**
 * @class MainWindow
 * @brief A class to represent the main window of the application.
 * @details This class inherits from QMainWindow and provides the main window of the application.
 * @see QMainWindow
 */
class MainWindow : public QMainWindow {
    Q_OBJECT

  public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void setupAnimation();

  private:
    /** @brief The UI object.*/
    Ui::MainWindow *ui;

    /** @brief The simulation engine.*/
    SimulationEngine *simulationEngine;

    /** @brief The list widget.*/
    QListWidget *listWidget;

    void initScene();
    void updateAnimation(); // Method to update the animation

protected:
    /** @brief The param widget.*/
    ParamWidget *paramWidget;

    /** @brief The expandable button widget.*/
    ExpandableButtonWidget *expandableWidget;

    /** @brief The overlay widget.*/
    OverlayWidget *overlay;

private slots:
    /**
     * @brief Slot to handle the save button click event.
     *
     * @return void
     */
    void saveSimulation();

    /**
     * @brief Overridden event filter method to handle key press events.
     *
     * @param object The object that the event is being filtered for
     * @param event The event that is being filtered
     * @return bool Whether the event was handled
     */
    bool eventFilter(QObject *object, QEvent *event) override;

    /**
     * @brief Overriden show event method to handle the show event.
     *
     * @param event The show event
     * @return void
     */
    void showEvent(QShowEvent *event) override;

    /**
     * @brief Overriden resize event method to handle the resize event.
     *
     * @param event The resize event
     * @return void
     */
    void resizeEvent(QResizeEvent *event) override;

    /**
     * @brief Overriden close event method to handle the close event.
     *
     * @param event The close event
     * @return void
     */
    void mouseDoubleClickEvent(QMouseEvent *event) override;

    /**
     * @brief Overriden mouse press event method to handle the mouse press event.
     *
     * @param event The mouse press event
     * @return void
     */
    void mousePressEvent(QMouseEvent *event) override;

    /**
     * @brief Overriden mouse move event method to handle the mouse move event.
     *
     * @param event The mouse move event
     * @return void
     */
    void mouseMoveEvent(QMouseEvent *event) override;

    /**
     * @brief Overriden mouse release event method to handle the mouse release event.
     *
     * @param event The mouse release event
     * @return void
     */
    void mouseReleaseEvent(QMouseEvent *event) override;

    /**
     * @brief Slot to handle the horizontal slider value changed event.
     *
     * @param value The new value of the slider
     * @return void
     */
    void on_horizontalSlider_valueChanged(int value);

    /**
     * @brief Slot to handle toggling the list.
     *
     * @return void
     */
    void toggleList();

    /**
     * @brief Slot to handle the item double click event from the list.
     *
     * @param item The item that was double clicked
     * @return void
     */
    void handleItemDoubleClick(QListWidgetItem *item);

    /**
     * @brief Slot to handle clear button click event.
     *
     * @return void
     */
    void on_pushButton_clicked();

    /**
     * @brief Slot to handle rotate anticlockwise button click event.
     *
     * @return void
     */
    void goLeft();

    /**
     * @brief Slot to handle stop rotating button click event.
     *
     * @return void
     */
    void stopRotating();

    /**
     * @brief Slot to handle rotate clockwise button click event.
     *
     * @return void
     */
    void goRight();

    /**
     * @brief Slot to handle move forward button click event.
     *
     * @return void
     */
    void goStraight();

    /**
     * @brief Slot to handle stop moving button click event.
     *
     * @return void
     */
    void stopMoving();
};
#endif // MAINWINDOW_H
```

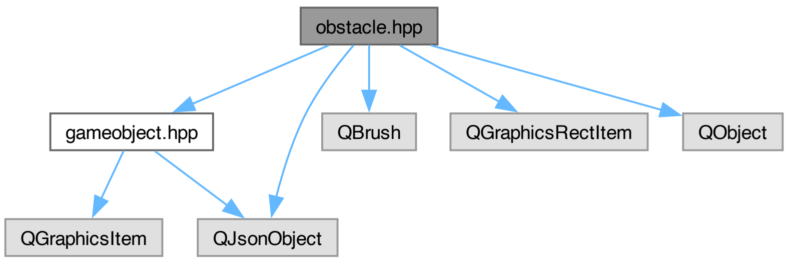
7.13 obstacle.hpp File Reference

This file contains the declaration of the Obstacle class.

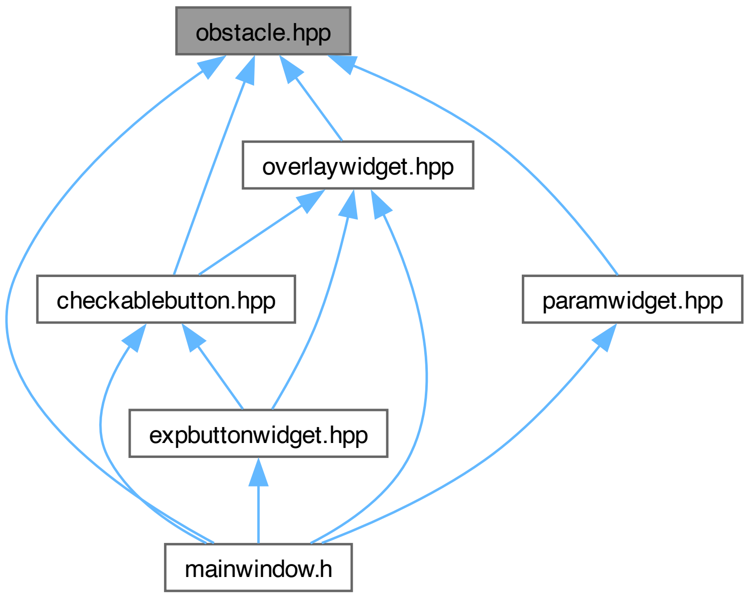
```
#include "gameobject.hpp"
#include <QBrush>
#include <QGraphicsRectItem>
#include <QJsonObject>
```

```
#include <QObject>
```

Include dependency graph for obstacle.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Obstacle

  A class to represent an obstacle.

7.13.1 Detailed Description

This file contains the declaration of the Obstacle class.

It is a subclass of the QGraphicsRectItem class and represents an obstacle in the game.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file obstacle.hpp.

7.14 obstacle.hpp

Go to the documentation of this file.

```
/**
 * @file obstacle.hpp
 * @brief This file contains the declaration of the Obstacle class.
 * @details It is a subclass of the QGraphicsRectItem class and represents an obstacle in the game.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef OBSTACLE_HPP
#define OBSTACLE_HPP

#include "gameobject.hpp"
#include <QBrush>
#include <QGraphicsRectItem>
#include <QJsonObject>
#include <QObject>

/**
 * @class Obstacle
 * @brief A class to represent an obstacle.
 * @details This class inherits from QGraphicsRectItem and GameObject. It represents an obstacle in a
game.
 */
class Obstacle : public QObject, public QGraphicsRectItem, public GameObject {
    Q_OBJECT

  public:
    /**
     * @brief Default constructor.
     * @param parent The parent QGraphicsItem.
     * @return void
     */
    Obstacle(QGraphicsItem *parent = nullptr);

    /**
     * @brief Copy constructor.
     * @param Obstacle The Obstacle object to copy.
     * @return void
     */
    Obstacle(const Obstacle &)
        : QGraphicsRectItem() {}

    /**
     * @brief Destructor.
     */
    ~Obstacle();

    /**
     * @brief Set the position of the obstacle.
     * @param x The x-coordinate of the position.
     * @param y The y-coordinate of the position.
     * @return void
     */
    void setPos(qreal x, qreal y) override;

    /**
     * @brief Set the rotation of the obstacle.
     * @param angle The angle of the rotation.
     * @return void
     */
    void setRotation(qreal angle) override;

    /**
```

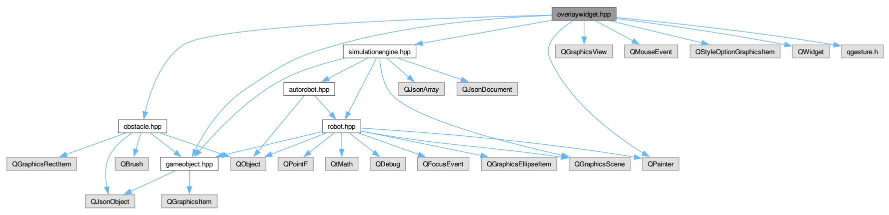
```
     * @brief Get the rotation of the obstacle.
     * @return The rotation of the obstacle as a qreal.
     */
    qreal rotation() override { return QGraphicsRectItem::rotation(); }

    /**
     * @brief Get the center of the obstacle.
     * @return The center of the obstacle as a QPointF.
     */
    QPointF getCenter() override { return boundingRect().center(); }

    /**
     * @brief Paint the obstacle.
     * @param painter Pointer to the QPainter object.
     * @param option Pointer to the QStyleOptionGraphicsItem object.
     * @param widget Pointer to the QWidget object.
     */
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;

    /**
     * @brief Get the position of the obstacle.
     * @return The position of the obstacle as a QPointF object.
     */
    QPointF getPos() override;

    /**
     * @brief Convert the obstacle to a JSON object.
     * @return The obstacle as a QJsonObject.
     */
    QJsonObject toJSON() override;

    /**
     * @brief Create an Obstacle object from a JSON object.
     * @param json The QJsonObject to convert.
     * @return A pointer to the created Obstacle object.
     */
    static Obstacle *fromJSON(const QJsonObject &json);

  signals:
    /**
     * @brief Signal emitted when the parameters of the obstacle are updated.
     * @return void
     */
    void paramsUpdated();

    /**
     * @brief Signal emitted when the obstacle is removed.
     * @return void
     */
    void obstacleSepuku();
};

#endif // OBSTACLE_HPP
```

7.15 overlaywidget.hpp File Reference

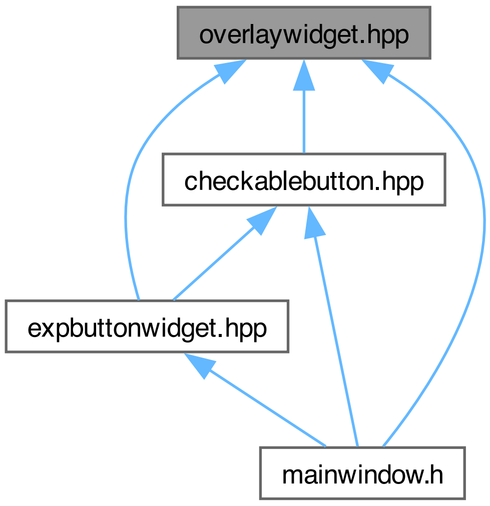
This file contains the declaration of the OverlayWidget class.

```
#include "gameobject.hpp"
#include "obstacle.hpp"
#include "simulationengine.hpp"
#include <QGraphicsView>
#include <QMouseEvent>
#include <QPainter>
#include <QStyleOptionGraphicsItem>
#include <QWidget>
#include <qgesture.h>
```

Include dependency graph for overlaywidget.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class OverlayWidget

  A class to represent an overlay widget.

7.15.1 Detailed Description

This file contains the declaration of the OverlayWidget class.

It is a subclass of the QWidget class and represents an overlay widget.

Authors

Tomáš Hobza, Jakub Všetečka

Definition in file overlaywidget.hpp.

7.16 overlaywidget.hpp

Go to the documentation of this file.

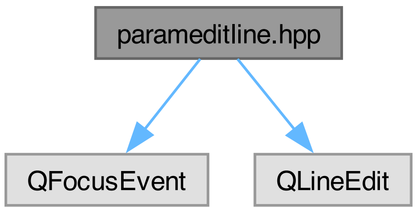
```
/**
 * @file overlaywidget.hpp
 * @brief This file contains the declaration of the OverlayWidget class.
 * @details It is a subclass of the QWidget class and represents an overlay widget.
 * @authors Tomáš Hobza, Jakub Všetečka
 */

#ifndef OVERLAYWIDGET_HPP
#define OVERLAYWIDGET_HPP

#include "gameobject.hpp"
#include "obstacle.hpp"
#include "simulationengine.hpp"
#include <QGraphicsView>
#include <QMouseEvent>
#include <QPainter>
#include <QStyleOptionGraphicsItem>
#include <QWidget>
#include <qgesture.h>

/**
 * @class OverlayWidget
 * @brief A class to represent an overlay widget.
 * @details This class provides an interface for creating and managing overlay widgets.
 * @see QWidget
 */
class OverlayWidget : public QWidget {
  public:
    /**
     * @brief Construct a new Overlay Widget object.
     *
     * @param parent The parent widget. Default is nullptr.
     * @param simEng The simulation engine. Default is nullptr.
     * @param graphView The graphics view. Default is nullptr.
     */
    explicit OverlayWidget(QWidget *parent = nullptr, SimulationEngine *simEng = nullptr,
QGraphicsView *graphView = nullptr);

    /**
     * @brief Try grab the object based on the mouse position.
     * @param event The mouse event.
     * @return void
     */
    void trySetSail(QMouseEvent *event);

    /**
     * @brief Drag the object based on the mouse position in the overlay.
     * @param event The mouse event.
     * @return void
     */
    void navigateTheSea(QMouseEvent *event);

    /**
     * @brief Anchor the object based on the mouse position back to scene.
     * @return void
     */
    void anchor();

    /**
     * @brief Get the time constant of the simulation engine.
     * @return qreal* The time constant of the simulation engine.
     */
    qreal *getTimeConstant() { return simEng->getTimeConstant(); }

    /**
     * @brief Get the active object.
     * @return GameObject* The active object.
     */
    void setActiveObject(GameObject *obj) { activeObject = obj; }

    /**
     * @brief Get the last mouse position.
     * @return QPoint The last mouse position.
     */
    void setLastMousePos(QPoint pos) { lastMousePos = pos; }

  protected:
    /** @brief The active object. */
    GameObject *activeObject;

    /** @brief The last mouse position. */
    QPoint lastMousePos;
    QPoint offset;

    /** @brief The simulation engine. */
    SimulationEngine *simEng;

    /** @brief The graphics view. */
    QGraphicsView *graphView;

    /** @brief The option for the graphics item. */
    QStyleOptionGraphicsItem option;

    /**
     * @brief Override the mousePressEvent method.
     * @param event The mouse event.
     * @return void
     */
    void paintEvent(QPaintEvent *event) override;

    /**
     * @brief Convert the point to the rotated system.
     * @param point The point in the scene.
     * @param angle The angle of the rotation.
     * @return QPoint The point in the rotated system.
     */
    QPoint convertToRotatedSystem(QPoint point, qreal angle);

    /**
     * @brief Convert the point from the rotated system.
     * @param point The point in the rotated system.
     * @param angle The angle of the rotation.
     * @return QPoint The point in the rotated system.
     */
    QPoint convertFromRotatedSystem(QPoint point, qreal angle);
};

#endif // OVERLAYWIDGET_HPP
```

7.17 parameditline.hpp File Reference

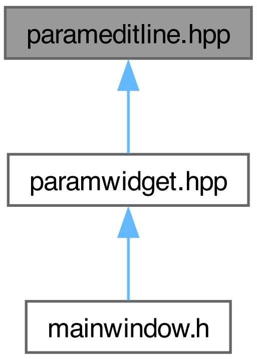
This file contains the declaration of the ParamEditLine class.

```
#include <QFocusEvent>
#include <QLineEdit>
```

Include dependency graph for parameditline.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ParamEditLine

  A class to represent a line edit widget for editing parameters.

7.17.1 Detailed Description

This file contains the declaration of the ParamEditLine class.

It is a subclass of the QLineEdit class and represents a line edit widget for editing parameters.

Authors

Tomáš Hobza, Jakub Všetečka

Date

03.05.2024

Definition in file parameditline.hpp.

7.18 parameditline.hpp

Go to the documentation of this file.

```
/**
 * @file parameditline.hpp
 * @brief This file contains the declaration of the ParamEditLine class.
 * @details It is a subclass of the QLineEdit class and represents a line edit widget for editing
parameters.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 03.05.2024
 */

#ifndef PARAMEDITLINE_HPP
#define PARAMEDITLINE_HPP

#include <QFocusEvent>
#include <QLineEdit>

/**
 * @class ParamEditLine
 * @brief A class to represent a line edit widget for editing parameters.
 * @details This class inherits from QLineEdit and provides a line edit widget for editing parameters.
 * @see QLineEdit
 */
class ParamEditLine : public QLineEdit {
    Q_OBJECT

  public:
    /**
     * @brief Default constructor.
     * @param parent The parent widget.
     */
    explicit ParamEditLine(QWidget *parent = nullptr)
        : QLineEdit(parent) {}

    /**
     * @brief Overridden focusInEvent method.
     * @param event The focus event.
     * @return void
     */
    void focusInEvent(QFocusEvent *event) override {
        QLineEdit::focusOutEvent(event);
        emit focusIn();
    }

    /**
     * @brief Overridden focusOutEvent method.
     * @param event The focus event.
     * @return void
     */
    void focusOutEvent(QFocusEvent *event) override {
        QLineEdit::focusOutEvent(event);
        emit focusOut();
    }

  signals:
    /**
     * @brief Signal emitted when the line edit widget gains focus.
     * @return void
     */
    void focusIn();

    /**
     * @brief Signal emitted when the line edit widget loses focus.
     * @return void
     */
    void focusOut();
};

#endif // PARAMEDITLINE_HPP
```

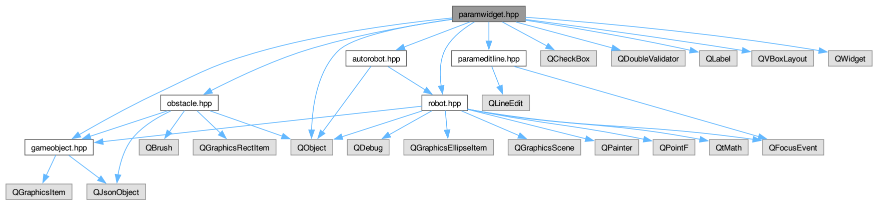
7.19 paramwidget.hpp File Reference

This file contains the declaration of the ParamWidget class.

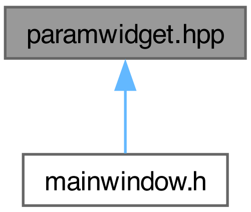
```
#include "autorobot.hpp"
#include "gameobject.hpp"
```

```
#include "obstacle.hpp"
#include "parameditline.hpp"
#include "robot.hpp"
#include <QCheckBox>
#include <QDoubleValidator>
#include <QLabel>
#include <QObject>
#include <QVBoxLayout>
#include <QWidget>
```

Include dependency graph for paramwidget.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ParamWidget

    A class to represent a widget for editing parameters of game objects.

7.19.1 Detailed Description

This file contains the declaration of the ParamWidget class.

It is a subclass of the QWidget class and represents a widget for editing parameters of game objects.

Authors

Tomáš Hobza, Jakub Všetečka

Date

03.05.2024

Definition in file paramwidget.hpp.

7.20 paramwidget.hpp

Go to the documentation of this file.

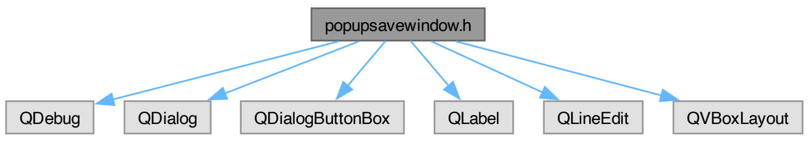
```
/**
 * @file paramwidget.hpp
 * @brief This file contains the declaration of the ParamWidget class.
 * @details It is a subclass of the QWidget class and represents a widget for editing parameters of
game objects.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 03.05.2024
 */

#ifndef PARAMWIDGET_HPP
#define PARAMWIDGET_HPP

#include "autorobot.hpp"
#include "gameobject.hpp"
#include "obstacle.hpp"
#include "parameditline.hpp"
#include "robot.hpp"
#include <QCheckBox>
#include <QDoubleValidator>
#include <QLabel>
#include <QObject>
#include <QVBoxLayout>
#include <QWidget>

/**
 * @class ParamWidget
 * @brief A class to represent a widget for editing parameters of game objects.
 * @details This class inherits from QWidget and provides a widget for editing parameters of game
objects.
 * @see QWidget
 */
class ParamWidget : public QWidget {
    Q_OBJECT

  public:
    /**
     * @brief Default constructor.
     * @param parent The parent widget.
     */
    explicit ParamWidget(QWidget *parent = nullptr);

    /**
     * @brief Set the game object whose parameters will be displayed.
     * @param object The game object.
     * @return void
     */
    void stalk(AutoRobot *robot);

    /**
     * @brief Set the game object whose parameters will be displayed.
     * @param object The game object.
     * @return void
     */
    void stalk(Obstacle *obstacle);

    /**
     * @brief Set the game object whose parameters will be displayed.
     * @param object The game object.
     * @return void
     */
    void stalk(Robot *robot);

    /**
     * @brief Stop editing the parameters of the game object.
     * @return void
     */
    void stopStalking();

  private:
    /** @brief The layout of the widget. */
    QVBoxLayout *layout;

    /** @brief The game object whose parameters are being displayed. */
    GameObject *stalkedObject = nullptr;

    /** @brief Whether the widget should keep updating the parameters of the game object. */
    bool keepUpdating = true;

    /** @brief The validator for the number input. */
    QDoubleValidator *numberValidator;

    /** @brief The labels and line edit widgets for editing the parameters. */
    QLabel *labelDetectionDistance;
    ParamEditLine *detectionDistance;
    QLabel *labelAngleToRotate;
    ParamEditLine *angleToRotate;
    QLabel *labelDirection;
    QCheckBox *direction;
    QLabel *labelSpeed;
    ParamEditLine *speed;
    QLabel *labelRadius;
    ParamEditLine *radius;
    QLabel *labelAngle;
    ParamEditLine *angle;
    QLabel *labelSize;
    ParamEditLine *size;

    /**
     * @brief Set up the line edit widget for editing a parameter.
     * @param lineEdit The line edit widget.
     * @param label The label for the line edit widget.
     * @return void
     */
    void setUpEditLine(ParamEditLine *lineEdit, QLabel *label);

    /**
     * @brief Show the parameters of the game object.
     * @param robot The robot whose parameters will be displayed.
     * @return void
     */
    void show(Robot *robot);

    /**
     * @brief Show the parameters of the game object.
     * @param robot The robot whose parameters will be displayed.
     * @return void
     */
    void show(AutoRobot *robot);

    /**
     * @brief Show the parameters of the game object.
     * @param obstacle The obstacle whose parameters will be displayed.
     * @return void
     */
    void show(Obstacle *obstacle);

    /**
     * @brief Hide the widget.
     * @return void
     */
    void hide();

    /**
     * @brief Disconnect the widget from the game object.
     * @return void
     */
    void disconnectStalkedObject();

  private slots:
    /**
     * @brief Signal to set the detection distance of the game object.
     * @return void
     */
    void setDetectionDistance();

    /**
     * @brief Signal to set the angle to rotate of the game object.
     * @return void
     */
    void setAngleToRotate();

    /**
     * @brief Signal to set the direction of the game object.
     * @return void
     */
    void setDirection();

    /**
     * @brief Signal to set the speed of the game object.
     * @return void
     */
    void setSpeed();

    /**
     * @brief Signal to set the radius of the game object.
     * @return void
     */
    void setRadius();

    /**
     * @brief Signal to set the angle of the game object.
     * @return void
     */
    void setAngle();

    /**
     * @brief Signal to set the size of the game object.
     * @return void
     */
    void setSize();

    /**
     * @brief Signal to update the parameters of the game object.
     * @return void
     */
    inline void focusIn() { keepUpdating = false; }

    /**
     * @brief Signal to update the parameters of the game object.
     * @return void
     */
    inline void focusOut() { keepUpdating = true; }

    /**
     * @brief Update the parameters of the game object.
     * @return void
     */
    void updateAutoRobot();

    /**
     * @brief Update the parameters of the game object.
     * @return void
     */
    void updateObstacle();

    /**
     * @brief Update the parameters of the game object.
     * @return void
     */
    void updateRobot();
};

#endif // PARAMWIDGET_HPP
```

7.21 popupsavewindow.h File Reference

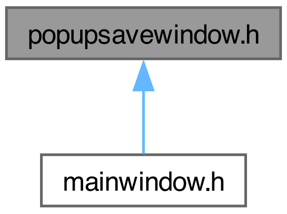
This file contains the declaration of the PopupSaveWindow class.

```
#include <QDebug>
#include <QDialog>
#include <QDialogButtonBox>
#include <QLabel>
#include <QLineEdit>
#include <QVBoxLayout>
```

Include dependency graph for popupsavewindow.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PopupSaveWindow

  A class to represent a popup save window.

7.21.1 Detailed Description

This file contains the declaration of the PopupSaveWindow class.

It is a subclass of the QDialog class and represents a popup save window.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file popupsavewindow.h.

7.22 popupsavewindow.h

Go to the documentation of this file.

```
/**
 * @file popupsavewindow.h
 * @brief This file contains the declaration of the PopupSaveWindow class.
 * @details It is a subclass of the QDialog class and represents a popup save window.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef POPUPSAVEWINDOW_H
#define POPUPSAVEWINDOW_H

#include <QDebug>
#include <QDialog>
#include <QDialogButtonBox>
#include <QLabel>
```

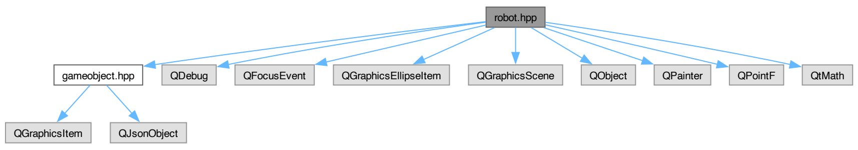
```
#include <QLineEdit>
#include <QVBoxLayout>

/**
 * @class PopupSaveWindow
 * @brief A class to represent a popup save window.
 * @details This class provides an interface for creating and managing a popup save window.
 * @see QDialog
 */
class PopupSaveWindow : public QDialog {
    Q_OBJECT

  public:
    /**
     * @brief Construct a new Popup Save Window object.
     * @param parent The parent widget. Default is nullptr.
     */
    explicit PopupSaveWindow(QWidget *parent = nullptr);
    ~PopupSaveWindow();

    /**
     * @brief Get the entered text.
     * @return QString The entered text.
     */
    QString getEnteredText() { return enteredText; }

  private:
    /** @brief The entered text. */
    QString enteredText;

    /** @brief The line edit widget. */
    QLineEdit *lineEdit;

  private slots:

    /**
     * @brief Slot to handle the ok button click event.
     * @return void
     */
    void onOkClicked();
};

#endif // POPUPSAVEWINDOW_H
```

7.23 robot.hpp File Reference

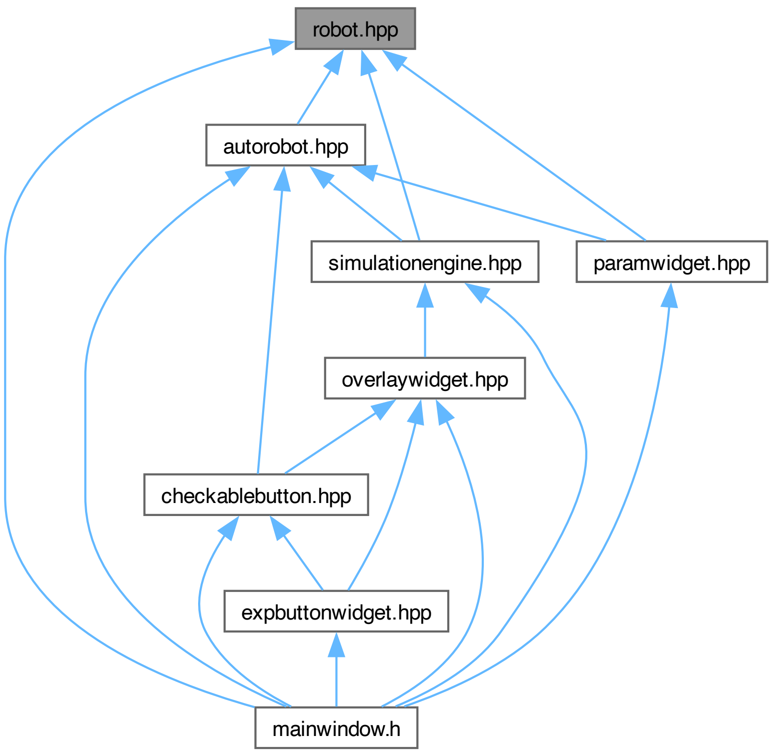
This file contains the declaration of the Robot class.

```
#include "gameobject.hpp"
#include <QDebug>
#include <QFocusEvent>
#include <QGraphicsEllipseItem>
#include <QGraphicsScene>
#include <QObject>
#include <QPainter>
#include <QPointF>
#include <QtMath>
```

Include dependency graph for robot.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Robot

    A class to represent a robot in the simulation. By default, the robot is a circle with a line drawn to represent its direction.

Macros

- #define BODY_COLLISION_MARGIN 1

7.23.1 Detailed Description

This file contains the declaration of the Robot class.

It is a subclass of the QGraphicsEllipseItem class and represents a robot in the simulation.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file robot.hpp.

7.23.2 Macro Definition Documentation

7.23.2.1 BODY_COLLISION_MARGIN

```
#define BODY_COLLISION_MARGIN 1
```

Definition at line 22 of file robot.hpp.

7.24 robot.hpp

Go to the documentation of this file.

```
00001 /**
00002  * @file robot.hpp
00003  * @brief This file contains the declaration of the Robot class.
00004  * @details It is a subclass of the QGraphicsEllipseItem class and represents a robot in the
      simulation.
00005  * @authors Tomáš Hobza, Jakub Všetečka
00006  * @date 02.05.2024
00007  */
00008 
00009 #ifndef ROBOT_HPP
00010 #define ROBOT_HPP
00011 
00012 #include "gameobject.hpp"
00013 #include <QDebug>
00014 #include <QFocusEvent>
00015 #include <QGraphicsEllipseItem>
00016 #include <QGraphicsScene>
00017 #include <QObject>
00018 #include <QPainter>
00019 #include <QPointF>
00020 #include <QtMath>
00021 
00022 #define BODY_COLLISION_MARGIN 1
00023 
00024 /**
00025  * @brief A class to represent a robot in the simulation. By default, the robot is a circle with a
      line drawn to represent its direction.
00026  */
00027 class Robot : public QObject, public QGraphicsEllipseItem, public GameObject {
00028     Q_OBJECT
00029 
00030   public:
00031     /**
00032      * @brief Enum to represent the direction of rotation of the robot.
00033      */
00034     enum RotationDirection {
00035         Left = -1, // Counter-clockwise
00036         None = 0,  // No rotation
00037         Right = 1  // Clockwise
00038     };
00039 
00040     enum { Type = QGraphicsItem::UserType + 1 };
00041 
00042     /**
00043      * @brief Default constructor.
00044      * @param parent The parent QGraphicsItem.
00045      * @param timeConstant The time constant of the simulation.
00046      * @return void
00047      * @details The time constant is used to calculate the speed of the robot.
00048      */
00049     Robot(QGraphicsItem *parent = nullptr, qreal *timeConstant = nullptr);
00050 
00051     ~Robot();
00052 
00053     /** Override the paint method to draw a line showing the direction of the robot */
00054     virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
      override;
00055 
00056     /** Override setPos to adjust to center-based positioning */
00057     void setPos(const QPointF &pos);
00058 
00059     /** Overload setPos to accept x and y coordinates */
00060     void setPos(qreal x, qreal y) override;
00061 
00062     /* Override the boundingRect method to adjust the bounding rectangle */
00063     virtual QRectF boundingRect() const override;
```

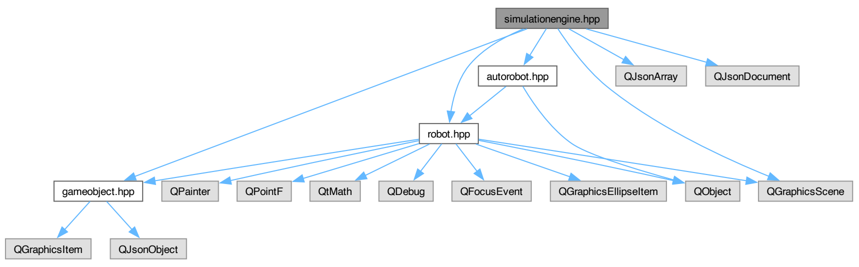
```

    /** Override pos to adjust to center-based positioning */
    QPointF pos();

    qreal getRadius() const;

    /**
     * @brief Set the move speed of the robot.
     *
     * @param speed
     */
    void setMoveSpeed(qreal speed);

    /**
     * @brief Get the move speed of the robot.
     *
     * @return qreal
     */
    qreal getMoveSpeed();

    /**
     * @brief Set the rotation speed of the robot.
     *
     * @param speed
     */
    void setRotationSpeed(qreal speed);

    /**
     * @brief Get the rotation speed of the robot.
     *
     * @return qreal
     */
    qreal getRotationSpeed();

    /**
     * @brief Allow the robot to be moved by setting the isMoving flag to true.
     */
    void startMoving();

    /**
     * @brief Stop the robot from moving by setting the isMoving flag to false.
     */
    void stopMoving();

    /**
     * @brief Start rotating the robot in the given direction.
     *
     * @param direction
     */
    void startRotating(RotationDirection direction);

    /**
     * @brief Stop the robot from rotating by setting the isRotating flag to None.
     */
    void stopRotating();

    /**
     * @brief Get the direction vector of the robot.
     *
     * @return `QPointF` - Normalized vector representing the direction of the robot on the x and y
axes
     */
    QPointF getDirectionVector();

    /**
     * @brief Check if the robot will collide with any other item in the scene or the scene boundaries
if it moves by the given vector.
     *
     * @param moveVector The vector by which the robot will move
     * @param allowAnticollision Flag to indicate if anticollision is allowed
     * @return `true` - if the robot will collide; `false` - if the robot will not collide
     */
    virtual bool willCollide(QPointF directionVector, qreal magnitude, bool allowAnticollision =
false);

    /**
     * @brief Move the robot based on its current direction and speed. Returns true if the robot
moved, false if it didn't (e.g. if it hit a boundary).
     *
     * @return true
     * @return false
     */
    virtual bool move();

    /**
     * @brief Get the type of the robot.
     * @return int
     */
    int type() const override { return Type; }

    /**
     * @brief Get the position of the robot.
     * @return QPointF
     */
    QPointF getPos() override;

    /**
     * @brief Convert the robot to a JSON object.
     * @return QJsonObject
     */
    virtual QJsonObject toJSON() override;

    /**
     * @brief Create a Robot object from a JSON object.
     * @param object The JSON object.
     * @param timeConstant The time constant of the simulation.
     * @return Robot*
     */
    static Robot *fromJSON(const QJsonObject &object, qreal *timeConstant);

    /**
     * @brief Toggle the active state of the robot.
     * @details If the robot is active, it will be drawn with a light gray fill. If it is inactive, it
 will be drawn with a transparent fill.
     * @return void
     */
    inline void toggleActive() {
        active = !active;
        active ? setBrush(QBrush(Qt::lightGray)) : setBrush(QBrush(Qt::transparent));
    }

    /**
     * @brief Check if the robot is active.
     * @return bool
     */
    inline bool isActive() { return active; }

    /**
     * @brief Get the angle of the robot.
     * @return qreal
     */
    qreal getAngle() { return rotation(); }

    /**
     * @brief Set the angle of the robot.
     * @param angle The angle to set.
     * @return void
     */
    void setRadius(qreal radius);

    /**
     * @brief Get the center of the robot.
     * @return QPointF
     */
    QPointF getCenter() override { return boundingRect().center(); }

    /**
     * @brief Get the time constant of the simulation.
     * @return qreal
     */
    qreal rotation() override {
        return QGraphicsEllipseItem::rotation();
    }

    /**
     * @brief Set the rotation of the robot.
     * @param angle The angle to set.
     * @return void
     */
    void setRotation(qreal angle) override {
        QGraphicsEllipseItem::setRotation(angle);
    }

  signals:

    /**
     * @brief Signal emitted when the parameters of the robot are updated.
     * @return void
     */
    void paramsUpdated();

    /**
     * @brief Signal emitted when the robot is removed.
     * @return void
     */
    void robotSepuku();

  protected:
    /** @brief The speed of the robot */
    qreal move_speed = 1;
    /** @brief The speed of the rotation of the robot */
    qreal rotation_speed = 1;

    /** @brief Flag to indicate if the robot is moving */
    bool isMoving = false;

    /** @brief Flag to indicate the direction of rotation */
    RotationDirection isRotating = RotationDirection::None;

    /** @brief The time constant of the simulation */
    qreal *timeConstant = nullptr;

    /** @brief The radius of the robot */
    void keyReleaseEvent(QKeyEvent *event);

    /**
     * @brief Overridden keyPressEvent method.
     * @details This method is called when a key is pressed while the robot is focused.
     * @param event The key event.
     * @return void
     */
    void keyPressEvent(QKeyEvent *event);

  private:
    /** @brief Flag to indicate if the robot is active */
    bool active = false;
};

#endif // ROBOT_HPP
```

7.25 simulationengine.hpp File Reference

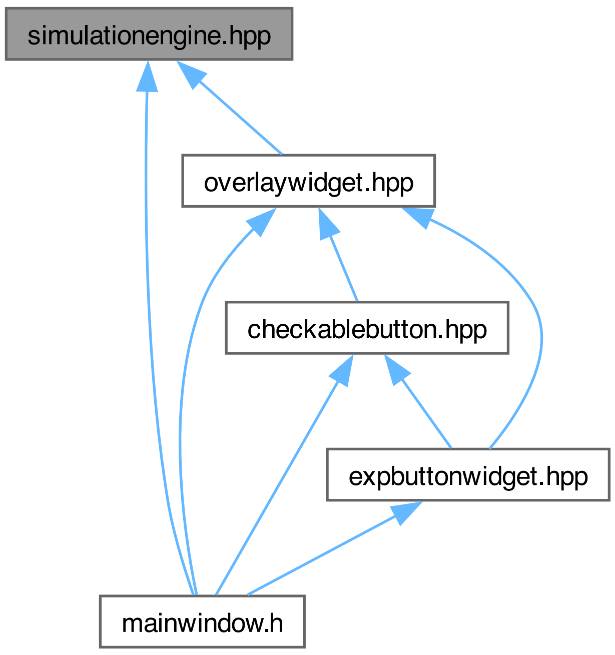
This file contains the declaration of the SimulationEngine class.

```
#include "autorobot.hpp"
#include "gameobject.hpp"
#include "robot.hpp"
#include <QGraphicsScene>
#include <QJsonArray>
#include <QJsonDocument>
```

Include dependency graph for simulationengine.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class SimulationEngine

7.25.1 Detailed Description

This file contains the declaration of the SimulationEngine class.

It is a subclass of the QGraphicsScene class and represents the simulation engine.

Authors

Tomáš Hobza, Jakub Všetečka

Date

02.05.2024

Definition in file simulationengine.hpp.

7.26 simulationengine.hpp

Go to the documentation of this file.

```
/**
 * @file simulationengine.hpp
 * @brief This file contains the declaration of the SimulationEngine class.
 * @details It is a subclass of the QGraphicsScene class and represents the simulation engine.
 * @authors Tomáš Hobza, Jakub Všetečka
 * @date 02.05.2024
 */

#ifndef SIMULATIONENGINE_H
#define SIMULATIONENGINE_H

#include "autorobot.hpp"
#include "gameobject.hpp"
#include "robot.hpp"
#include <QGraphicsScene>
#include <QJsonArray>
#include <QJsonDocument>

class SimulationEngine : public QGraphicsScene {
  public:
    SimulationEngine(QObject *parent = nullptr, int fps = 60, qreal simulationSpeed = 1.0 / 16.0);

    ~SimulationEngine();

    /**
     * @brief Simulation Frames-Per-Second getter.
     * @return int
     */
    int getFPS();

    /**
     * @brief Get the time it takes to render a single frame.
     * @return int
     */
    int getFrameTime();

    /**
     * @brief Set the simulation Frames-Per-Second.
     * @param fps
     */
    void setFPS(int fps);

    /**
     * @brief Get the simulation speed.
     * @return qreal
     */
    qreal getSimulationSpeed();

    /**
     * @brief Set the simulation speed.
     * @param speed
     * @return void
     */
    void setSimulationSpeed(qreal speed);

    /**
     * @brief Update the time constant.
     * @return void
     */
    void updateTimeConstant();

    /**
     * @brief Get the time constant pointer.
     * @return qreal*
     */
    qreal *getTimeConstant();

    /**
     * @brief Check if a point is inside the scene.
     * @param point
     * @return bool
     */
    bool isInsideScene(const QPointF &point) const;

    /**
     * @brief Get the robot that is currently being controlled.
     * @return Robot*
     */
    Robot *getControlledRobot();

    /**
     * @brief Set the robot that is currently being controlled.
     * @param robot
     * @return void
     */
    void setControlledRobot(Robot *robot);

    /**
     * @brief Save the simulation.
     * @param filename The name of the file to save the simulation to.
     * @details The file will be saved in the JSON format in folder "simulations"
     * @return void
     */
    bool saveSimulation(const QString &filename = "simulation");

    /**
     * @brief Load the simulation.
     * @param filename The name of the file to load the simulation from.
     * @details The file will be loaded from the JSON format from folders "simulations" and "exmaples"
     * @return void
     */
    bool loadSimulation(QString filename = "simulation");

    /**
     * @brief Read the simulation from a JSON object.
     * @param json The JSON object to read.
     * @return void
     */
    void read(const QJsonObject &json);

    /**
     * @brief Convert the simulation to a JSON object.
     * @return QJsonObject
     */
    QJsonObject toJson() const;

    /**
     * @brief Clear the scene.
     *
     */
    void clearScene();

  private:
    /** The frames per second of the simulation engine. */
    int fps = 60;
    /** The speed of the simulation engine. */
    qreal simulationSpeed = 1;

    /** The time constant of the simulation engine. */
    qreal timeConstant = 1;

    /** The robot that is currently being controlled. */
    Robot *controlledRobot = nullptr;
};

#endif // SIMULATIONENGINE_H
```

Index