Turtle Graphics

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

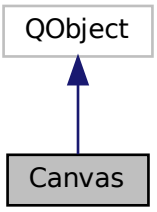
Class Documentation

3.1 Canvas Class Reference

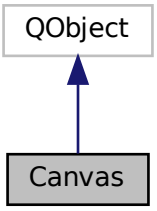
This class represents a canvas where obstacles can be generated and managed.

`#include <canvas.hpp>`

Inheritance diagram for Canvas:



Collaboration diagram for Canvas:



Signals

- void obstacles_changed ()

  Signal emitted when the list of obstacles has changed.
- void width_changed ()

  Signal emitted when the width of the canvas has changed.
- void height_changed ()

  Signal emitted when the height of the canvas has changed.

Public Member Functions

- Canvas (QObject *parent=nullptr)

  Constructs a Canvas object with optional parent.
- ~Canvas ()

  Destructor for the Canvas object, which clears all obstacles.
- qreal width () const

  Returns the width of the canvas.
- qreal height () const

  Returns the height of the canvas.
- void set_width (qreal width)

  Sets the width of the canvas and notifies the change.
- void set_height (qreal height)

  Sets the height of the canvas and notifies the change.
- QPolygonF get_shape () const

  Returns the shape of the canvas.
- Q_INVOKABLE void generate_obstacles (int count, const QPointF &turtle_pos)

  Generates a specified number of random obstacles while avoiding overlap with a turtle.
- Q_INVOKABLE void clear_obstacles ()

  Clears all obstacles from the canvas.
- Q_INVOKABLE QVariantList get_obstacle_points (int index) const

  Retrieves the points of a specific obstacle.
- Q_INVOKABLE QString get_obstacle_color (int index) const

  Retrieves the color of a specific obstacle.
- int obstacle_count () const

  Returns the total number of obstacles currently in the canvas.
- const QVector< Obstacle * > & get_obstacles () const

  Returns a reference to the list of obstacles.

Properties

- QML_ELEMENTint obstacle_count

  The total number of obstacles in the canvas.
- qreal width

  The width of the canvas.
- qreal height

  The height of the canvas.

3.1.1 Detailed Description

This class represents a canvas where obstacles can be generated and managed.

This class allows for obstacle generation, clearing, and access to the obstacle count and properties. The canvas size is adjustable, and the obstacles are generated randomly while avoiding overlap with a turtle.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Canvas()

```
Canvas::Canvas (
            QObject * parent = nullptr )  [explicit]
```

Constructs a Canvas object with optional parent.

Parameters

parent	The parent QObject, defaults to nullptr.

3.1.3 Member Function Documentation

3.1.3.1 generate_obstacles()

```
void Canvas::generate_obstacles (
            int count,
            const QPointF & turtle_pos )
```

Generates a specified number of random obstacles while avoiding overlap with a turtle.

Parameters

count	The number of obstacles to generate.
turtle_pos	The position of the turtle.

3.1.3.2 get_obstacle_color()

```
QString Canvas::get_obstacle_color (
            int index ) const
```

Retrieves the color of a specific obstacle.

Parameters

index	The index of the obstacle.

Returns

A QString representing the color of the obstacle in hexadecimal format.

3.1.3.3 get_obstacle_points()

```
QVariantList Canvas::get_obstacle_points (
            int index ) const
```

Retrieves the points of a specific obstacle.

Parameters

index	The index of the obstacle.

Returns

A QVariantList of the obstacle's points.

3.1.3.4 get_obstacles()

```
const QVector<Obstacle*>& Canvas::get_obstacles ( ) const  [inline]
```

Returns a reference to the list of obstacles.

Returns

A reference to the vector of obstacles.

3.1.3.5 get_shape()

```
QPolygonF Canvas::get_shape ( ) const
```

Returns the shape of the canvas.

Returns

The shape of the canvas, QPolygonF with the origin at (0,0) and 4 points forming a rectangle.

3.1.3.6 height()

```
qreal Canvas::height ( ) const  [inline]
```

Returns the height of the canvas.

Returns

The height of the canvas.

3.1.3.7 obstacle_count()

```
int Canvas::obstacle_count ( ) const  [inline]
```

Returns the total number of obstacles currently in the canvas.

Returns

The number of obstacles.

3.1.3.8 set_height()

```
void Canvas::set_height (
            qreal height )
```

Sets the height of the canvas and notifies the change.

Parameters

height	The new height of the canvas.

3.1.3.9 set_width()

```
void Canvas::set_width (
            qreal width )
```

Sets the width of the canvas and notifies the change.

Parameters

width	The new width of the canvas.

3.1.3.10 width()

```
qreal Canvas::width ( ) const  [inline]
```

Returns the width of the canvas.

Returns

The width of the canvas.

3.1.4 Property Documentation

3.1.4.1 height

```
qreal Canvas::height  [read], [write]
```

The height of the canvas.

This property can be read and written. Changes are notified through the height_changed signal.

3.1.4.2 obstacle_count

```
QML_ELEMENTint Canvas::obstacle_count  [read]
```

The total number of obstacles in the canvas.

This property is read-only and notifies when the list of obstacles changes.

3.1.4.3 width

```
qreal Canvas::width  [read], [write]
```

The width of the canvas.

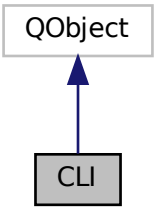
This property can be read and written. Changes are notified through the width_changed signal.

3.2 CLI Class Reference

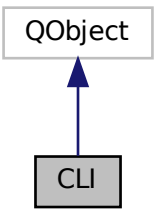
The CLI class provides an interface for processing commands, managing history, and interacting with a parser and state management.

`#include <CLI.hpp>`

Inheritance diagram for CLI:



Collaboration diagram for CLI:



Signals

- void commandProcessed (const QString &message)

  Emitted when a command has been processed.
- void outputChanged ()

  Emitted when the output log changes.
- void requestQuit ()

  Emitted to request the application to quit.

Public Member Functions

- CLI (QObject *parent=nullptr)

  Constructor for the CLI class.
- Q_INVOKABLE void setParser (Parser *parser)

  Sets the parser instance to process commands.
- Q_INVOKABLE void processCommand (const QString &command)

  Processes a given command by parsing and executing it.
- Q_INVOKABLE QString getOutput () const

  Retrieves the current output log as a single string.
- Q_INVOKABLE void clearOutput ()

  Clears the output log.
- Q_INVOKABLE QStringList getCommandHistory () const

  Retrieves the current command history as a QStringList.
- Q_INVOKABLE void loadScript (const QString &filename)

  Loads and processes a script file by passing its contents to the parser.
- void appendToOutputLog (const QString &message)

  Appends a message to the output log and emits the appropriate signals.

3.2.1 Detailed Description

The CLI class provides an interface for processing commands, managing history, and interacting with a parser and state management.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CLI()

```
CLI::CLI (
            QObject * parent = nullptr )  [explicit]
```

Constructor for the CLI class.

Parameters

parent	Pointer to the parent QObject (optional).

3.2.3 Member Function Documentation

3.2.3.1 appendToOutputLog()

```
void CLI::appendToOutputLog (
            const QString & message )
```

Appends a message to the output log and emits the appropriate signals.

Parameters

message	The message to append to the output log.

3.2.3.2 clearOutput()

```
void CLI::clearOutput ( )
```

Clears the output log.

Note

This function is callable from QML.

3.2.3.3 commandProcessed

```
void CLI::commandProcessed (
            const QString & message )  [signal]
```

Emitted when a command has been processed.

Parameters

message	A QString containing the processed command or result.

3.2.3.4 getCommandHistory()

```
QStringList CLI::getCommandHistory ( ) const
```

Retrieves the current command history as a QStringList.

Returns

A QStringList containing all commands in the history.

Note

This function is callable from QML.

3.2.3.5 getOutput()

```
QString CLI::getOutput ( ) const
```

Retrieves the current output log as a single string.

Returns

A QString containing the concatenated output log.

Note

This function is callable from QML.

3.2.3.6 loadScript()

```
void CLI::loadScript (
            const QString & filename )
```

Loads and processes a script file by passing its contents to the parser.

This method opens the specified script file, reads its contents, and passes an ifstream to the parser for processing.

Parameters

filename	The filename of the script to be loaded.

3.2.3.7 processCommand()

```
void CLI::processCommand (
            const QString & command )
```

Processes a given command by parsing and executing it.

Parameters

command	The command string to process.

Note

This function is callable from QML.

3.2.3.8 setParser()

```
void CLI::setParser (
            Parser * parser )
```

Sets the parser instance to process commands.

Parameters

parser	Pointer to the Parser object.

Note

This function is callable from QML.

3.3 Line Struct Reference

A structure representing a line segment with customizable attributes.

`#include <turtlecontrol.h>`

Public Member Functions

- **Line** (const QPointF &start=QPointF(), const QPointF &end=QPointF(), const QColor &color=QColor(), float width=1.f)

Public Attributes

- QPointF start_

  The starting point of the line segment.
- QPointF end_

  The ending point of the line segment.
- QColor color_

  The color of the line.
- float width_

  The width of the line.

Properties

- Q_GADGETQPointF **start**
- QPointF **end**
- QColor **color**
- float **width**

3.3.1 Detailed Description

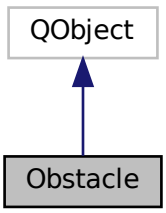
A structure representing a line segment with customizable attributes.

3.4 Obstacle Class Reference

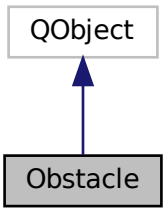
The class represents an obstacle in the canvas,.

`#include <obstacle.hpp>`

Inheritance diagram for Obstacle:



Collaboration diagram for Obstacle:



Signals

- void points_changed ()

  Emitted when the points of the obstacle change.
- void color_changed ()

  Emitted when the color of the obstacle changes.
- void position_changed ()

  Emitted when the position of the obstacle changes.

Public Member Functions

- Obstacle (QObject *parent=nullptr)

  Default constructor for Obstacle.
- Obstacle (const QPolygonF &points, const QColor &color, QObject *parent=nullptr)

  Constructs an Obstacle with specific properties.
- QPolygonF get_points () const

  Gets the points defining the obstacle's shape.
- QColor get_color () const

  Gets the color of the obstacle.
- QPointF get_position () const

  Gets the position (center point) of the obstacle.
- float get_bounding_radius () const

  Gets the bounding radius of the obstacle. Gets the distance from the center to the furthest point of the obstacle's bounding rect. Used in collision calculations.
- void set_points (const QPolygonF &points)

  Sets the points defining the obstacle's shape.
- void set_color (const QColor &color)

  Sets the color of the obstacle.
- void set_position (const QPointF &pos)

  Sets the position (center point) of the obstacle.
- bool intersects (const QRectF &rect) const

  Checks if the obstacle intersects with a given rectangle.

Properties

- QML_ELEMENTQPolygonF points

  Points of the obstacle.
- QColor color

  Color of the obstacle.
- QPointF position

  Position of the obstacle.

3.4.1 Detailed Description

The class represents an obstacle in the canvas,.

This class encapsulates the points, color, and position of an obstacle, and provides methods to manipulate these properties. It also includes functionality for checking whether the obstacle intersects with a given rectangle.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Obstacle() [1/2]

```
Obstacle::Obstacle (
            QObject * parent = nullptr )  [explicit]
```

Default constructor for Obstacle.

Constructs an obstacle with default properties (empty polygon, red color).

Parameters

parent	The parent object, default is nullptr.

3.4.2.2 Obstacle() [2/2]

```
Obstacle::Obstacle (
            const QPolygonF & points,
            const QColor & color,
            QObject * parent = nullptr )
```

Constructs an Obstacle with specific properties.

Parameters

points	The points defining the obstacle's shape.
color	The color of the obstacle.
parent	The parent object, default is nullptr.

3.4.3 Member Function Documentation

3.4.3.1 get_bounding_radius()

```
float Obstacle::get_bounding_radius ( ) const  [inline]
```

Gets the bounding radius of the obstacle. Gets the distance from the center to the furthest point of the obstacle's bounding rect. Used in collision calculations.

Returns

The bounding radius.

3.4.3.2 get_color()

```
QColor Obstacle::get_color ( ) const  [inline]
```

Gets the color of the obstacle.

Returns

The color of the obstacle as a QColor.

3.4.3.3 get_points()

```
QPolygonF Obstacle::get_points ( ) const  [inline]
```

Gets the points defining the obstacle's shape.

Returns

The points of the obstacle as a QPolygonF.

3.4.3.4 get_position()

```
QPointF Obstacle::get_position ( ) const  [inline]
```

Gets the position (center point) of the obstacle.

Returns

The position as a QPointF.

3.4.3.5 intersects()

```
bool Obstacle::intersects (
            const QRectF & rect ) const
```

Checks if the obstacle intersects with a given rectangle.

This is done by checking whether the bounding box of the obstacle intersects the given rectangle.

Parameters

rect	The rectangle to check for intersection.

Returns

True if the obstacle intersects the rectangle, false otherwise.

3.4.3.6 set_color()

```
void Obstacle::set_color (
            const QColor & color )
```

Sets the color of the obstacle.

Parameters

color	The new color for the obstacle.

3.4.3.7 set_points()

```
void Obstacle::set_points (
            const QPolygonF & points )
```

Sets the points defining the obstacle's shape.

This will update the obstacle's position based on the new points.

Parameters

points	The new points for the obstacle.

3.4.3.8 set_position()

```
void Obstacle::set_position (
            const QPointF & pos )
```

Sets the position (center point) of the obstacle.

This will translate the points of the obstacle to maintain the shape, but move it to the new position.

Parameters

pos	The new position for the obstacle.

3.4.4 Property Documentation

3.4.4.1 color

```
QColor Obstacle::color  [read], [write]
```

Color of the obstacle.

The color used to draw the obstacle.

3.4.4.2 points

```
QML_ELEMENTQPolygonF Obstacle::points  [read], [write]
```

Points of the obstacle.

A polygon representing the vertices of the obstacle shape.

3.4.4.3 position

```
QPointF Obstacle::position  [read], [write]
```

Position of the obstacle.

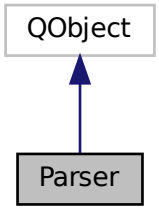
The center point of the obstacle in the scene.

3.5 Parser Class Reference

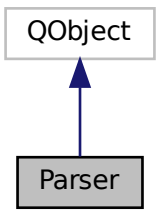
A class to parse and execute commands lines/scripts for controlling a Turtle object.

```
#include <parser.hpp>
```

Inheritance diagram for Parser:



Collaboration diagram for Parser:



Public Slots

- void animation_done ()

  Slot to process the end of an animation.

Signals

- void forward (float distance)

  Signal to move the turtle forward.
- void turn (float angle)

  Signal to turn the turtle by an angle.
- void up ()

  Signal to lift the turtle's pen to stop drawing.
- void down ()

  Signal to lower the turtle's pen to continue drawing.
- void setpos (QPointF pos)

  Signal to set the turtle's position.
- void setrot (float rot)

  Signal to set the turtle's rotation.
- void setsize (float size)

  Signal to set the turtle's pen size.
- void setspeed (float speed)

  Signal to set the turtle's movement speed.
- void arc (float radius, float angle)

  Signal to draw an arc.

Public Member Functions

- Parser (QObject ∗parent=nullptr)

  Constructs a Parser object.
- Q_INVOKABLE std::vector< std::string > parse_line (const QString &inputQ)

  Parses a single line of input and sends the commands to the Turtle.
- Q_INVOKABLE std::vector< std::string > parse_script (std::ifstream &file)

  Parses an entire script file and executes the commands.

3.5.1 Detailed Description

A class to parse and execute commands lines/scripts for controlling a Turtle object.

The Parser class has two key functions for parsing single command lines or entire scripts To execute commands, a corresponding Qt signal is sent to the Turtle module.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Parser()

```
Parser::Parser (
            QObject * parent = nullptr )  [explicit]
```

Constructs a Parser object.

Parameters

parent	Optional QObject parent.

3.5.3 Member Function Documentation

3.5.3.1 arc

```
void Parser::arc (
            float radius,
            float angle )  [signal]
```

Signal to draw an arc.

Parameters

radius	The radius of the arc.
angle	The angle to sweep in degrees.

3.5.3.2 forward

```
void Parser::forward (
            float distance )  [signal]
```

Signal to move the turtle forward.

Parameters

distance	The distance to move forward.

3.5.3.3 parse_line()

```
std::vector< std::string > Parser::parse_line (
            const QString & inputQ )
```

Parses a single line of input and sends the commands to the Turtle.

Parameters

inputQ	The input line as a QString.

Returns

Vector of commands that were successfully parsed and executed.

3.5.3.4 parse_script()

```
std::vector< std::string > Parser::parse_script (
            std::ifstream & file )
```

Parses an entire script file and executes the commands.

Parameters

file	The input script file as an ifstream.

Returns

Vector of commands that were successfully parsed and executed.

3.5.3.5 setpos

```
void Parser::setpos (
            QPointF pos )  [signal]
```

Signal to set the turtle's position.

Parameters

pos	The new position as a QPointF.

3.5.3.6 setrot

```
void Parser::setrot (
            float rot )  [signal]
```

Signal to set the turtle's rotation.

Parameters

rot	The new rotation angle in degrees.

3.5.3.7 setsize

```
void Parser::setsize (
            float size )  [signal]
```

Signal to set the turtle's pen size.

Parameters

size	The new pen size.

3.5.3.8 setspeed

```
void Parser::setspeed (
            float speed )  [signal]
```

Signal to set the turtle's movement speed.

Parameters

speed	The new movement speed.

3.5.3.9 turn

```
void Parser::turn (
            float angle )  [signal]
```

Signal to turn the turtle by an angle.

Parameters

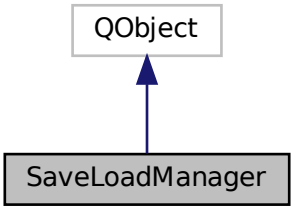
angle	The angle to turn in degrees.

3.6 SaveLoadManager Class Reference

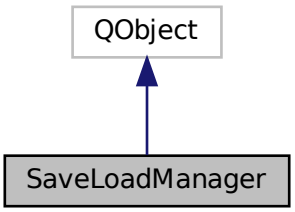
Manages saving and loading of application state, screenshots, and related data.

```
#include <SaveLoadManager.hpp>
```

Inheritance diagram for SaveLoadManager:



Collaboration diagram for SaveLoadManager:



Signals

- void mainWindowChanged ()

  Signal emitted when the main window reference changes.
- void buildFolderChanged ()

  Signal emitted when the build folder path changes.

Public Member Functions

- SaveLoadManager (QObject ∗parent=nullptr)

  Constructs a SaveLoadManager object.
- QObject ∗ mainWindow () const

  Gets the main window reference.
- QString buildFolder () const

  Gets the build folder path.
- Q_INVOKABLE void setMainWindow (QObject ∗mainWindow)

  Sets the main window reference.
- Q_INVOKABLE void setTurtleControl (TurtleControl ∗turtleControl)

  Sets the TurtleControl reference.

- Q_INVOKABLE void setCLI (CLI *cli)

  Sets the CLI reference.
- Q_INVOKABLE void setBuildFolder (const QString &buildFolder)

  Sets the build folder path.
- Q_INVOKABLE void saveScreenshot ()

  Saves a screenshot to the specified file.
- Q_INVOKABLE void saveState (const QString &fileName)

  Saves the current state to the specified file.
- Q_INVOKABLE void loadState (const QString &filePath)

  Loads a state from the specified file.

Properties

- QObject * mainWindow

  The main window reference.
- QString buildFolder

  The build folder path.

3.6.1 Detailed Description

Manages saving and loading of application state, screenshots, and related data.

This class provides functions to save screenshots, application state, and load states. It exposes relevant properties and functions to QML.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 SaveLoadManager()

```
SaveLoadManager::SaveLoadManager (
            QObject * parent = nullptr )  [explicit]
```

Constructs a SaveLoadManager object.

Parameters

parent	The parent QObject. Defaults to nullptr.

3.6.3 Member Function Documentation

3.6.3.1 buildFolder()

```
QString SaveLoadManager::buildFolder ( ) const
```

Gets the build folder path.

Returns

The build folder path as a QString.

3.6.3.2 loadState()

```
void SaveLoadManager::loadState (
            const QString & filePath )
```

Loads a state from the specified file.

Parameters

filePath	The path of the file to load the state from.

This function is Q_INVOKABLE to allow QML access.

3.6.3.3 mainWindow()

```
QObject * SaveLoadManager::mainWindow ( ) const
```

Gets the main window reference.

Returns

A pointer to the main window object.

3.6.3.4 saveScreenshot()

```
void SaveLoadManager::saveScreenshot ( )
```

Saves a screenshot to the specified file.

Parameters

none.	This function is Q_INVOKABLE to allow QML access.

3.6.3.5 saveState()

```
void SaveLoadManager::saveState (
            const QString & fileName )
```

Saves the current state to the specified file.

Parameters

fileName	The name of the file to save the state to.

This function is Q_INVOKABLE to allow QML access.

3.6.3.6 setBuildFolder()

```
void SaveLoadManager::setBuildFolder (
            const QString & buildFolder )
```

Sets the build folder path.

Parameters

buildFolder	The new build folder path as a QString.

This function is Q_INVOKABLE to allow QML access.

3.6.3.7 setCLI()

```
void SaveLoadManager::setCLI (
            CLI * cli )
```

Sets the CLI reference.

Parameters

cli	A pointer to the CLI object.

This function is Q_INVOKABLE to allow QML access.

3.6.3.8 setMainWindow()

```
void SaveLoadManager::setMainWindow (
            QObject * mainWindow )
```

Sets the main window reference.

Parameters

mainWindow	A pointer to the main window object.

This function is Q_INVOKABLE to allow QML access.

3.6.3.9 setTurtleControl()

```
void SaveLoadManager::setTurtleControl (
            TurtleControl * turtleControl )
```

Sets the TurtleControl reference.

Parameters

turtleControl	A pointer to the TurtleControl object.

This function is Q_INVOKABLE to allow QML access.

3.6.4 Property Documentation

3.6.4.1 buildFolder

```
QString SaveLoadManager::buildFolder  [read], [write]
```

The build folder path.

This property holds the path to the folder used for saving and loading operations. It is accessible from QML.

3.6.4.2 mainWindow

```
QObject* SaveLoadManager::mainWindow  [read], [write]
```

The main window reference.

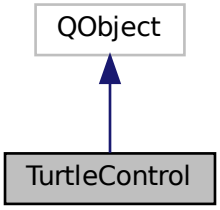
This property holds a reference to the main window object and is accessible from QML.

3.7 TurtleControl Class Reference

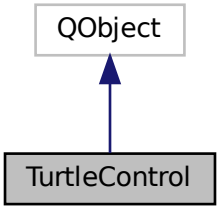
The TurtleControl class.

`#include <turtlecontrol.h>`

Inheritance diagram for TurtleControl:



Collaboration diagram for TurtleControl:



Public Slots

- void on_clicked ()

  Slot to handle click events. Triggers a random movement action.
- void set_speed (float speed)

  Sets the movement speed of the turtle.
- void set_position (QPointF position)

  Sets the turtle's position.
- void set_rotation (float rotation)

  Sets the turtle's rotation.
- void set_pen_down (bool b_pen_down)

  Sets the pen down state.
- void set_pen_radius (float radius)

Sets the pen radius.

- float turn (float degrees=30.f)

  Rotates the turtle clockwise by a specified number of degrees.
- float forward (float distance=100.f)

  Moves the turtle forward by a specified distance.
- float arc (float radius, float degrees=360.f)

  Draws an arc with the turtle.
- void set_lines (const QVector< Line > &lines)

  Replaces the current set of lines and notifies listeners.
- QVector< Line > get_lines () const

  Public function to get lines.
- void reset_state ()

  Resets the turtle to its initial state.

Signals

- void position_changed ()

  Emitted when the position of the turtle changes.
- void rotation_changed ()

  Emitted when the rotation of the turtle changes.
- void pen_down_changed ()

  Emitted when the pen state changes.
- void pen_radius_changed ()

  Emitted when the pen radius changes.
- void pen_color_changed ()

  Emitted when the pen color changes.
- void lines_changed ()

  Emitted when the lines vector changes.
- void on_movement_completed (MovementResult movement_result)

  Emitted when a movement operation is completed.
- void on_collision (QObject *hit_object, const QPolygonF &hit_polygon)

  Emitted when on collision event. Emitted when a collision with obstacle or the canvas borders occurs.

Public Member Functions

- TurtleControl (QObject *parent=nullptr)

  Constructs a TurtleControl object.
- QPointF position () const

  Gets the current position of the turtle.
- float rotation () const

  Gets the current rotation of the turtle.
- bool pen_down () const

  Checks if the pen is down.
- float pen_radius () const

  Gets the current pen radius.
- QColor pen_color () const

  Gets the current pen color.
- float get_speed () const

  Gets the current movement speed of the turtle.

- float get_arc_segments () const

  Gets the current Gets the current number arc segments. Number of segments needed to draw a full circle using the arc command.
- void set_arc_segments (float arc_segments)

  Sets arc segments. Sets the number of segments needed to draw a full circle using the arc command.
- int line_count () const

  Gets the total number of lines drawn by the turtle.
- QPointF get_forward_vector () const

  Calculates the forward vector based on the turtle's current rotation.
- QPointF get_right_vector () const

  Calculates the right vector based on the turtle's current rotation.
- const QPolygonF & get_shape () const

  Returns the shape of the cursor. Returns a circle with the turtle origin and a radius equal to the pen radius.
- void update_shape (const QPointF &translation_vector=QPointF())

  Used to update the turtle shape on position and pen changes.
- bool is_moving () const

  Indicates if the turtle is currently moving.
- Q_INVOKABLE void set_pen_color (const QColor &color)

  Sets the pen color.
- Q_INVOKABLE Line get_line (int index) const

  Retrieves a specific line from the collection.
- Q_INVOKABLE void set_canvas (Canvas *canvas)

  Sets the canvas.

Properties

- QML_ELEMENTQPointF position

  The current position of the turtle.
- float rotation

  The current rotation angle of the turtle in degrees.
- bool pen_down

  Indicates whether the pen is currently down.
- float pen_radius

  The radius of the pen.
- QColor pen_color

  The color of the pen.
- int line_count

  The total number of lines drawn by the turtle.

3.7.1 Detailed Description

The TurtleControl class.

The TurtleControl class handles the movement and pen properties of a turtle. It is designed to be used as a QML element, allowing for easy integration into QML-based applications. The class provides properties for position, rotation, pen state, radius, and color, along with methods to control the turtle's movement.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 TurtleControl()

```
TurtleControl::TurtleControl (
            QObject * parent = nullptr )  [explicit]
```

Constructs a TurtleControl object.

This constructor initializes the turtle's properties with default values.

Parameters

parent	Pointer to the parent QObject. This is used for managing the object hierarchy in Qt.

3.7.3 Member Function Documentation

3.7.3.1 arc

```
float TurtleControl::arc (
            float radius,
            float degrees = 360.f )  [slot]
```

Draws an arc with the turtle.

Parameters

radius	The radius of the arc.
degrees	The angle in degrees to draw the arc. Default value is 360 indicating a full circle.

Returns

The duration of the arc movement.

3.7.3.2 forward

```
float TurtleControl::forward (
            float distance = 100.f )  [slot]
```

Moves the turtle forward by a specified distance.

Parameters

distance	The distance to move the turtle forward. Default value is 100.

Returns

The movement duration.

3.7.3.3 get_arc_segments()

```
float TurtleControl::get_arc_segments ( ) const  [inline]
```

Gets the current Gets the current number arc segments. Number of segments needed to draw a full circle using the arc command.

Returns

The current number of arc segments.

3.7.3.4 get_forward_vector()

```
QPointF TurtleControl::get_forward_vector ( ) const
```

Calculates the forward vector based on the turtle's current rotation.

Returns

The forward vector as a QPointF.

3.7.3.5 get_line()

```
Line TurtleControl::get_line (
            int index ) const
```

Retrieves a specific line from the collection.

Parameters

index	The index of the line to retrieve.

Returns

The line at the given index, or an empty line if the index is invalid.

3.7.3.6 get_lines

```
QVector< Line > TurtleControl::get_lines ( ) const  [slot]
```

Public function to get lines.

Parameters

none.	

3.7.3.7 get_right_vector()

```
QPointF TurtleControl::get_right_vector ( ) const
```

Calculates the right vector based on the turtle's current rotation.

Returns

The right vector as a QPointF.

3.7.3.8 get_shape()

```
const QPolygonF& TurtleControl::get_shape ( ) const  [inline]
```

Returns the shape of the cursor. Returns a circle with the turtle origin and a radius equal to the pen radius.

Returns

The circular shape as a QPolygonF.

3.7.3.9 get_speed()

```
float TurtleControl::get_speed ( ) const  [inline]
```

Gets the current movement speed of the turtle.

Returns

The movement speed.

3.7.3.10 is_moving()

```
bool TurtleControl::is_moving ( ) const
```

Indicates if the turtle is currently moving.

Returns

True if the turtle movement animation exists and is being played, false otherwise.

3.7.3.11 line_count()

```
int TurtleControl::line_count ( ) const  [inline]
```

Gets the total number of lines drawn by the turtle.

Returns

The number of lines.

3.7.3.12 on_collision

```
void TurtleControl::on_collision (
            QObject * hit_object,
            const QPolygonF & hit_polygon )  [signal]
```

Emitted when on collision event. Emitted when a collision with obstacle or the canvas borders occurs.

Parameters

hit_object	The object collided with.
hit_polygon	The shape of the collided object.

3.7.3.13 on_movement_completed

```
void TurtleControl::on_movement_completed (
            MovementResult movement_result )  [signal]
```

Emitted when a movement operation is completed.

Parameters

movement_result	The result of the movement operation.

3.7.3.14 pen_color()

```
QColor TurtleControl::pen_color ( ) const  [inline]
```

Gets the current pen color.

Returns

The current pen color as a QColor.

3.7.3.15 pen_down()

```
bool TurtleControl::pen_down ( ) const  [inline]
```

Checks if the pen is down.

Returns

True if the pen is down, false otherwise.

3.7.3.16 pen_radius()

```
float TurtleControl::pen_radius ( ) const  [inline]
```

Gets the current pen radius.

Returns

The current pen radius.

3.7.3.17 position()

```
QPointF TurtleControl::position ( ) const  [inline]
```

Gets the current position of the turtle.

Returns

The current position as a QPointF.

3.7.3.18 reset_state

```
void TurtleControl::reset_state ( )  [slot]
```

Resets the turtle to its initial state.

Sets the position to (450, 450) and clears the lines vector.

3.7.3.19 rotation()

```
float TurtleControl::rotation ( ) const  [inline]
```

Gets the current rotation of the turtle.

Returns

The current rotation angle in degrees.

3.7.3.20 set_arc_segments()

```
void TurtleControl::set_arc_segments (
            float arc_segments )  [inline]
```

Sets arc segments. Sets the number of segments needed to draw a full circle using the arc command.

Parameters

arc_segments	The new number of segments.

3.7.3.21 set_canvas()

```
Q_INVOKABLE void TurtleControl::set_canvas (
            Canvas * canvas )  [inline]
```

Sets the canvas.

Parameters

canvas	The pointer to the canvas.

3.7.3.22 set_lines

```
void TurtleControl::set_lines (
            const QVector< Line > & lines )  [slot]
```

Replaces the current set of lines and notifies listeners.

Parameters

lines	A QVector of Line structures representing the new lines.

3.7.3.23 set_pen_color()

```
void TurtleControl::set_pen_color (
            const QColor & color )
```

Sets the pen color.

Parameters

color	The new color for the pen.

3.7.3.24 set_pen_down

```
void TurtleControl::set_pen_down (
            bool b_pen_down )  [slot]
```

Sets the pen down state.

Parameters

b_pen_down	True to set the pen down, false to lift it.

3.7.3.25 set_pen_radius

```
void TurtleControl::set_pen_radius (
            float radius )  [slot]
```

Sets the pen radius.

Parameters

radius	The new radius for the pen.

3.7.3.26 set_position

```
void TurtleControl::set_position (
            QPointF position )  [slot]
```

Sets the turtle's position.

Parameters

position	The new position for the turtle.

3.7.3.27 set_rotation

```
void TurtleControl::set_rotation (
            float rotation )  [slot]
```

Sets the turtle's rotation.

Parameters

rotation	The new rotation angle in degrees.

3.7.3.28 set_speed

```
void TurtleControl::set_speed (
            float speed )  [inline], [slot]
```

Sets the movement speed of the turtle.

Parameters

speed	The new movement speed.

3.7.3.29 turn

```
float TurtleControl::turn (
            float degrees = 30.f )  [slot]
```

Rotates the turtle clockwise by a specified number of degrees.

Parameters

degrees	The angle in degrees to rotate the turtle. Default value is 30.

Returns

The movement duration.

3.7.3.30 update_shape()

```
void TurtleControl::update_shape (
            const QPointF & translation_vector = QPointF() )
```

Used to update the turtle shape on position and pen changes.

Parameters

translation_vector	Translation vector. Default is (0,0).

Index