Team Bionics

Line Robots

Table of Contents

Team Bionics **1**

Table of Contents **2**

What is the Project? **3**

What Line Robots is meant to accomplish **3**

Main features of Line Robots **4**

Line Robots quality attributes **5**

How end users can install Line Robots **6**

- Steps to compile the project from an executable: On macOS 6
- Steps to compile the project from an executable: On Windows OS 6

How end users can install build, run the software **6**

- Qt Installation Tutorial 6
- Running the Program 9

How end users can use the line robots application. **11**

Line Robots architecture and design **12**

- Class Diagram for Line Robots 14
- Sequence Diagram for Line Robots 15

Line Robots design pattern(s) and in which modules/components is used. **16**

The final state/condition of the Line Robots **17**

- Current Status of Line Robots 17
- Bugs and Issues 17
- Product Backlog: To be implemented 17

A explanation of the general file structure of the Line Robots **17**

Future Development and
Contribution of Line Robots **20**

What is the Project?

Team Bionics decided to design and incorporate a simulation of non-colliding line robots. The main overview of the project requirements that were requested by the customer, Ravi were the following:

1. Implement robots that travel along lines.
2. Lines should be path-like and constrain the movement of robots.
3. Robots should be able to have a maximum speed set individually.
4. Robots should be able to vary their speed up to their own maximum speed in order to avoid collisions.
5. Robots should be able to stop to allow robots on crossing paths to cross without collisions.
6. The lines and robots should be customizable by the user..

What Line Robots is meant to accomplish

Team Bionics accepted the challenge and designed a graphical user interface (GUI) application that matched the requirements of the Customer and more. This is an overview of the final product starting with the application used to build the software.

Line Robots application is a multi platform application that can be compiled and run on almost any modern computer. The front end of the application is an open source software known as Qt. Qt is a cross-platform application development framework for desktop, embedded and mobile. Supported Platforms include Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS and others. Due to extensive compatibility with a vast amount of architecture Team Bionics decided to use it as a front end framework.

Qt is not a programming language on its own. It is a framework written in C++. Therefore, C++ is the backend of the Line Robots application and the core of all the processes.

How Qt works is that it uses a preprocessor, known as the MOC (Meta-Object Compiler), to extend the C++ language with features like signals and slots. Additionally, Qt provides cross-platform graphics capabilities that use OS-native widgets. Thus, right before the compilation step, the MOC parses the source files written in Qt-extended C++ and generates standard compliant C++ sources from them. Thus the framework itself and applications/libraries using it can be compiled by any standard compliant C++ compiler like GCC.

Main features of Line Robots

Line Robots as of the most current release is a fully functional application built around the Customer's requests. Line

Line Robots Features:

- Fully Function GUI, designed with highest precision.
- The UI is built on Team Bionics core quality attributes: Usability, Integrity and Flexibility.
- The user is able to drag and drop directional lines on canvas.
    - Included Directions are:
        - North (up)
        - South (down)
        - East (right)
        - West (left)
- The user is able to drag and drop robots on the lines with four styles to choose from.
    - Included Designs are:
        - Triangle
        - Diamond
        - Circle
        - Square
- The user is able to set the speed of the robot.
- The user has a choice of 15 levels of speed to choose from.
    - He can tune the speed the moment the robot attaches to the line.
- The user has the ability to customize the robots by changing their color using the color picker.
- The user has the ability to register into memory a preset color of his choice.
- The user has the ability to set the hue, value, and saturation of the robot.
- The user has the ability to adjust the speed of the whole simulation.
- The user can “Run”, “Pause” or “Edit” the simulation with a click on a button.

- For convenience and ease, the user has the ability to “Undo” or “Redo” any choice that is made for the simulation.
- The user has the ability to “Clear” the whole canvas and start over.
- The user has the ability to exit the program at any time.
- The Robot is able to detect over robots on the line using the state of the art “Radar”, that can detect the proximity and adjust the speed based on the robot in front.
- The Robot is able to detect over robots in the intersections and yields to the other robots that are faster than it.
- The Line Robot is able to move based on the user's choice of direction of the “Line”.
- The Line Robots is a multi platform application.

Line Robots quality attributes

The main goal of Line Robots was to provide a piece of software that contains the quality and the usability of a well built application. Team Bionics core quality attributes which remained for the duration of the development process were: Usability, Integrity and Flexibility.

The center of the Line Robots is usability. The question that was asked every time a feature was implemented was “Can a person use the software correctly with minimal effort?” Even in the initial stages of development when the paper prototypes were designed the goal was to make a piece of software that will be intuitive and logical. When a user opens the program they will be able to identify the components and would not need to go into the documentation to figure out how to use it.

Line Robots satisfies the usability attribute because all of the controls are at the users disposal the moment they launch the program. In addition, all the features such as adding a line or a robot to the canvas uses an intuitive method of dragging the object to where the user wants it to be, making the software very easy to use.

Line Robots satisfies the integrity attribute because every single precaution and attention to detail was enforced to make sure that the software would not get into a bad state. The main goal in the development was to squash bugs as they came along. Even after the program was ready for release, Line Robots was modified several times to make sure that no unauthorized access could alter the code. This was done in two ways. One, the program was fortified with validations that prevent the user from doing something else with the program from what it was intended to do in the first place. Second, a multiplatform executable was designed. This increased the security, allowing any modern antivirus to scan the program and increased the usability of the program even further. The executable allowed the user to run the program without the need for compiling the code.

Line Robots satisfied the flexibility attribute because the software can be used in many different ways. As a development team one of the questions that was asked was: “How many different ways can users use this piece of software?”. To extend the flexibility of the program the main focus was to modularize the program as much as possible. This included the ability to take one piece or one function and modify it with ease. Furthermore, classes and a support factory were designed to support the basic functions of the program such as the line drag and drop. Thus, increasing the flexibility of the program even further.

How end users can install Line Robots

Steps to compile the project from an executable: On macOS

1. Navigate to the macOS folder and open the line_robots.dmg OR Extract to “macOS.zip” folder.
2. Extract the folder.
3. **Right Click on line_robots.app file.
4. Select “Open”
5. Select “Cancel”
6. **Right Click on line_robots.app file.
7. Click “Open”

Steps to compile the project from an executable: On Windows OS

(make sure you have Microsoft Visual C++ 2015 Redistributable or higher installed.)
Link: https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads

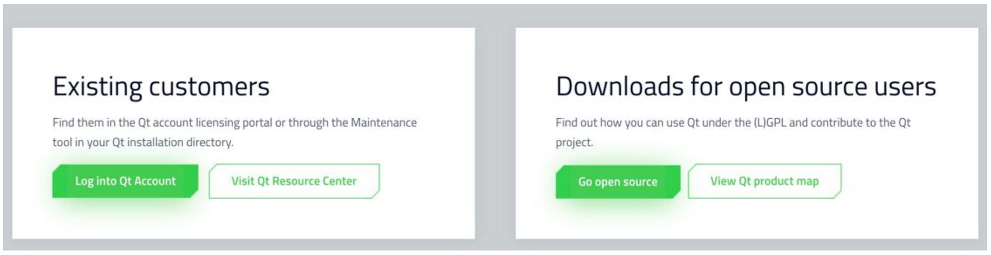
1. Extract to “WindowsVersion.zip” folder.
2. Click on line_robots.exe file.
3. Select “Open”

How end users can install build, run the software

Qt Installation Tutorial

1. Download Qt from https://www.qt.io/download

2. Download the Qt where it states “Downloads for open source users”



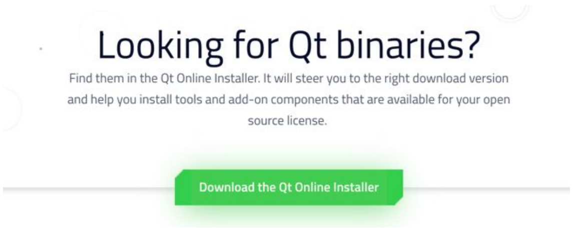
Existing customers

Find them in the Qt account licensing portal or through the Maintenance tool in your Qt installation directory.

Downloads for open source users

Find out how you can use Qt under the (L)GPL and contribute to the Qt project.

3. Scroll down and click on “Download the Qt Online Installer”

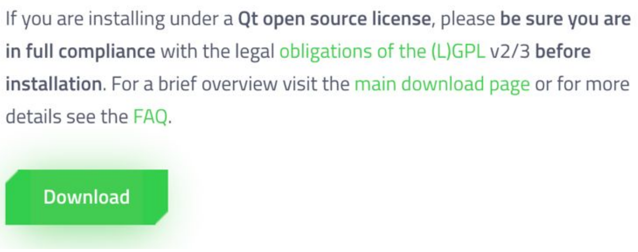


Looking for Qt binaries?

Find them in the Qt Online Installer. It will steer you to the right download version and help you install tools and add-on components that are available for your open source license.

A. The Browser will automatically detect the operating system.

B. Click “Download” button.



If you are installing under a **Qt open source license**, please **be sure you are in full compliance** with the legal obligations of the (L)GPL v2/3 **before installation**. For a brief overview visit the main download page or for more details see the FAQ.

4. Installation

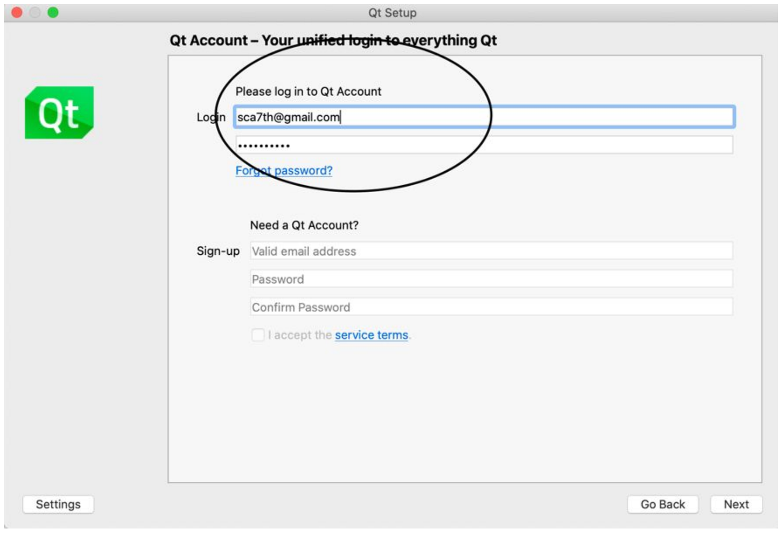
A. On macOS/Windows (Prerequisites: Have latest version of XCode installed)

a. Download qt-unified-mac-x64-3.2.2-online.dmg or qt-unified-windows-x86-3.2.2-online.exe

b. Double Click to install

c. Create an account for Qt (program will ask this due to being open source.

d. Verify in your email, then go back to the installation and login.



Qt Setup

Qt Account – Your unified login to everything Qt

Please log in to Qt Account

Login: sca7th@gmail.com

••••••••••

Forgot password?

Need a Qt Account?

Sign-up: Valid email address

Password

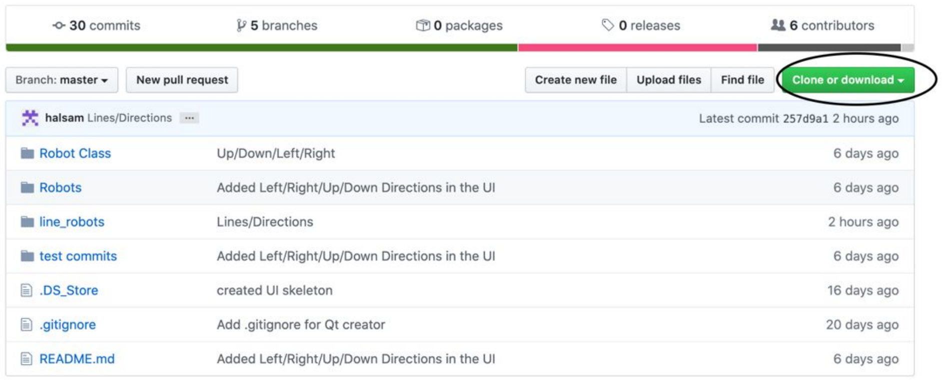
Confirm Password

I accept the service terms.

e. Qt will authenticate the login, then will ask for user to agree to the terms. Place a checkmark and agree.

f. Press “continue” twice.
g. Choose if you want to contribute to Qt, “press continue”
    i. select Qt 5.14.2 then click next.
    ii. agree to the terms
    iii. Set shortcut and continue.
h. Choose an installation directory for Qt. Then press “Continue”
j. Finish the installation.

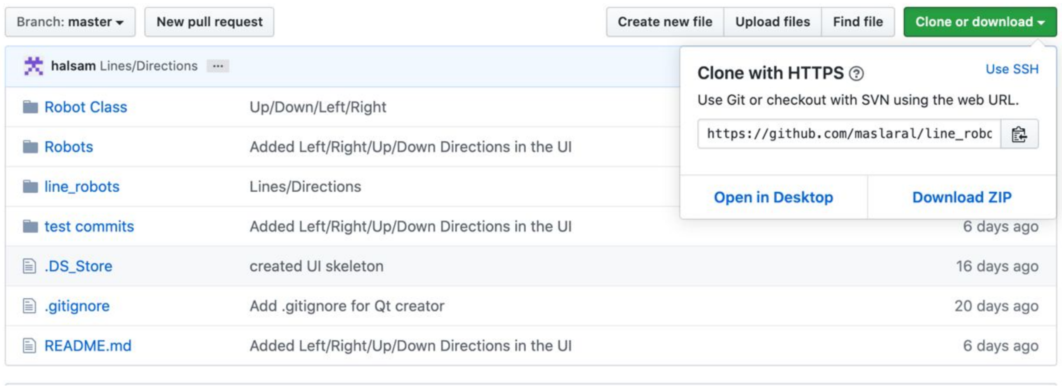
5. Download source from GitHub
    A. https://github.com/maslaral/line_robots
    B. Download source from GitHub, by clicking on the “Clone or Download”,



halsam Lines/Directions … Latest commit 257d9a1 2 hours ago

Robot Class	Up/Down/Left/Right	6 days ago
Robots	Added Left/Right/Up/Down Directions in the UI	6 days ago
line_robots	Lines/Directions	2 hours ago
test commits	Added Left/Right/Up/Down Directions in the UI	6 days ago
.DS_Store	created UI skeleton	16 days ago
.gitignore	Add .gitignore for Qt creator	20 days ago
README.md	Added Left/Right/Up/Down Directions in the UI	6 days ago

    C. “Download Zip” the extract the zip to any directory which would be easily accessible. (have 7zip, WinRAR or any unarchiving software to extract the files. Built in software can be used also)



Use Git or checkout with SVN using the web URL.

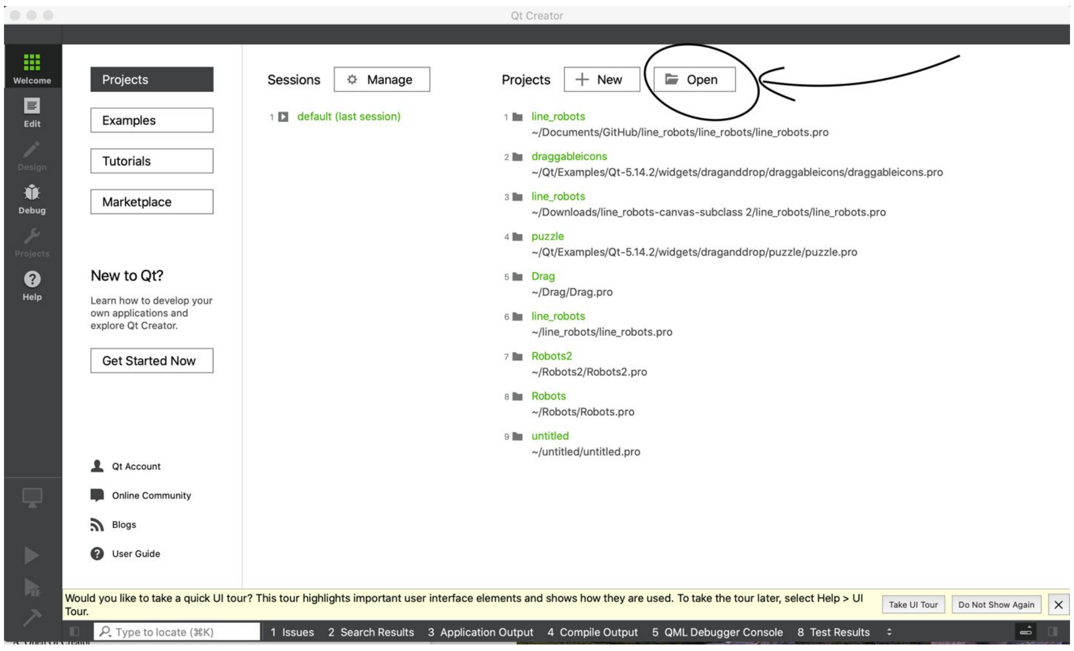
```
https://github.com/maslaral/line_robc
```

halsam Lines/Directions …

Robot Class	Up/Down/Left/Right	
Robots	Added Left/Right/Up/Down Directions in the UI	
line_robots	Lines/Directions	
test commits	Added Left/Right/Up/Down Directions in the UI	6 days ago
.DS_Store	created UI skeleton	16 days ago
.gitignore	Add .gitignore for Qt creator	20 days ago
README.md	Added Left/Right/Up/Down Directions in the UI	6 days ago

Running the Program

6. Running the Program
    A. Open Qt Creator
    B. Click “Open”

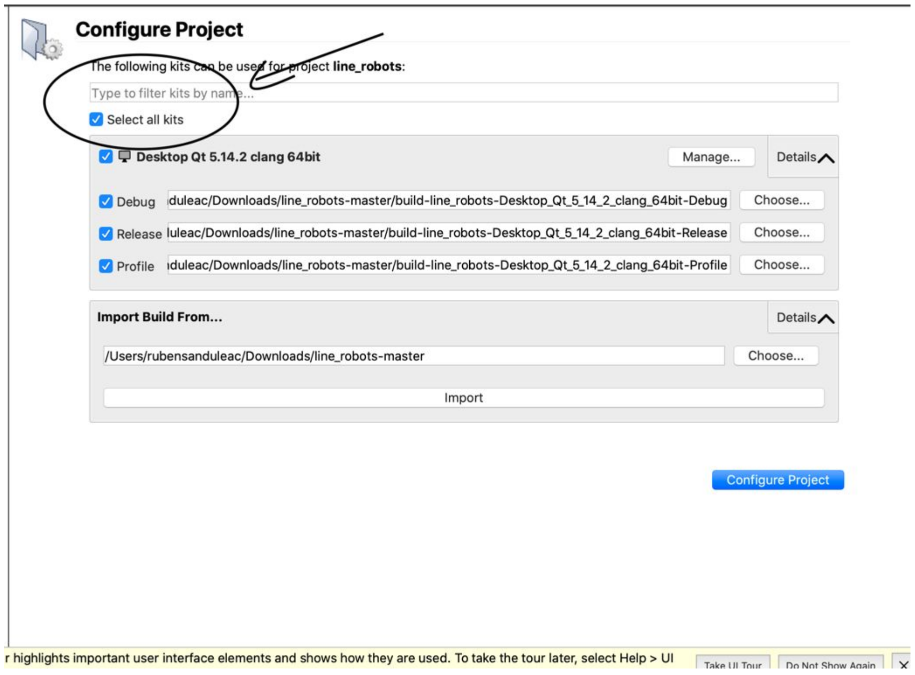


1 line_robots
~/Documents/GitHub/line_robots/line_robots/line_robots.pro
2 draggableicons
~/Qt/Examples/Qt-5.14.2/widgets/draganddrop/draggableicons/draggableicons.pro
3 line_robots
~/Downloads/line_robots-canvas-subclass 2/line_robots/line_robots.pro
4 puzzle
~/Qt/Examples/Qt-5.14.2/widgets/draganddrop/puzzle/puzzle.pro
5 Drag
~/Drag/Drag.pro
6 line_robots
~/line_robots/line_robots.pro
7 Robots2
~/Robots2/Robots2.pro
8 Robots
~/Robots/Robots.pro
9 untitled
~/untitled/untitled.pro

Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.

C. Navigate to the unarchived files to the folder and select “line_robots-master” □ “line_robots” □ line_robots.pro file.

D. Click Open.
E. An error might occur, disregard it click “ok”
F. Make sure “Select all kits” is checked


Configure Project

The following kits can be used for project **line_robots**:

Type to filter kits by name...

☑ Select all kits

☑ **Desktop Qt 5.14.2 clang 64bit**

☑ Debug	duleac/Downloads/line_robots-master/build-line_robots-Desktop_Qt_5_14_2_clang_64bit-Debug	Choose...
☑ Release	luleac/Downloads/line_robots-master/build-line_robots-Desktop_Qt_5_14_2_clang_64bit-Release	Choose...
☑ Profile	duleac/Downloads/line_robots-master/build-line_robots-Desktop_Qt_5_14_2_clang_64bit-Profile	Choose...

Import Build From...

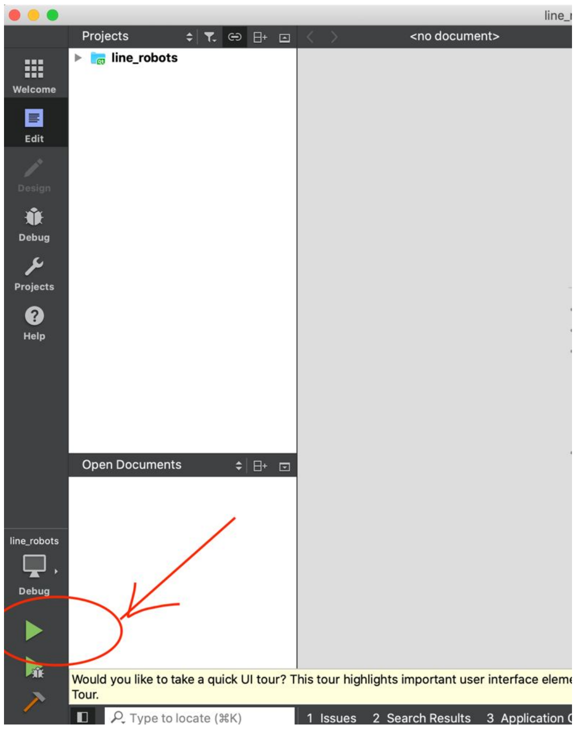
/Users/rubensanduleac/Downloads/line_robots-master

r highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI

G. Press Configure Project

7. Running the program

A. Press the play button



Would you like to take a quick UI tour? This tour highlights important user interface eleme
Tour.

B. The program will then run and will look like this:

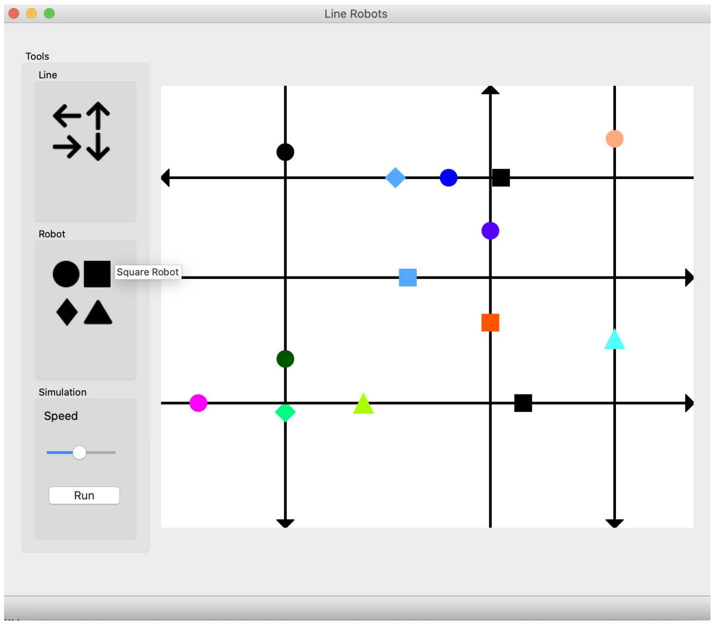


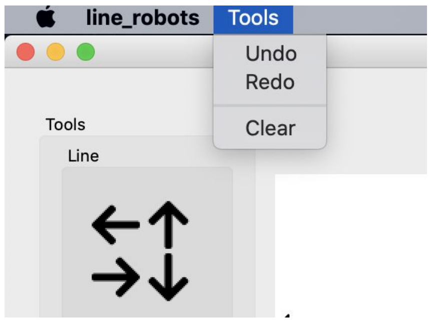
Figure 1A

How end users can use the line robots application.

1. Running the simulation:
    a. Drag a line anywhere on the canvas.
        i. Line can be dragged to the white canvas. Lines correspond to the direction where line faces. The movement of the robot will be in the direction the arrow points to.
    b. Select the Shape of the Robot and Drag the robot to the canvas to be placed on top of the line
        i. A speed prompt will be asked for the user to input.
            1. (Range: 1 – 15)
        ii. Set the speed (cannot be changed after the robot is set on the line.)
            1. Type the integer or use the arrow keys.
        iii. Adjust the color of the robot using the color picker or type the RGB code to select a specific color. Then press "**Ok**".
        iv. Place any additional robot or lines. If not, move to the next step.
    c. Press the "**Run**" Button to start the simulation.
        i. Once the "**Run**" is pressed no edits can be made until the "Pause" button is pressed.
    d. The robot will move from the starting position of the line to end position
        i. The robot will loop infinitely on that line.
        ii. The robot will slow down if the speed of movement is faster than the robot in front.
        iii. The robot will slow down and or Stop and the intersection if another robot is at the intersection.

e. Press the “**Pause**” button to temporarily pause the simulation, to place more lines or robots.
f. Use the **slider** to adjust the speed of the whole simulation.
g. To reset the canvas:
    i. On the navigation bar, go to “**Tools**” and press the “**Clear**” button.
h. To undo the last action:
    i. On the navigation bar, go to “**Tools**” and press the “**Undo**” button.
i. To redo the last action:
    i. On the navigation bar, go to “**Tools**” and press the “**Redo**” button.

2. Press the “**Exit**” button to exit the program.



Line Robots architecture and design

An overview of your software’s architecture and design

The Line Robots design follows the three principles, single responsibility, open/closed and interface segregation principle.

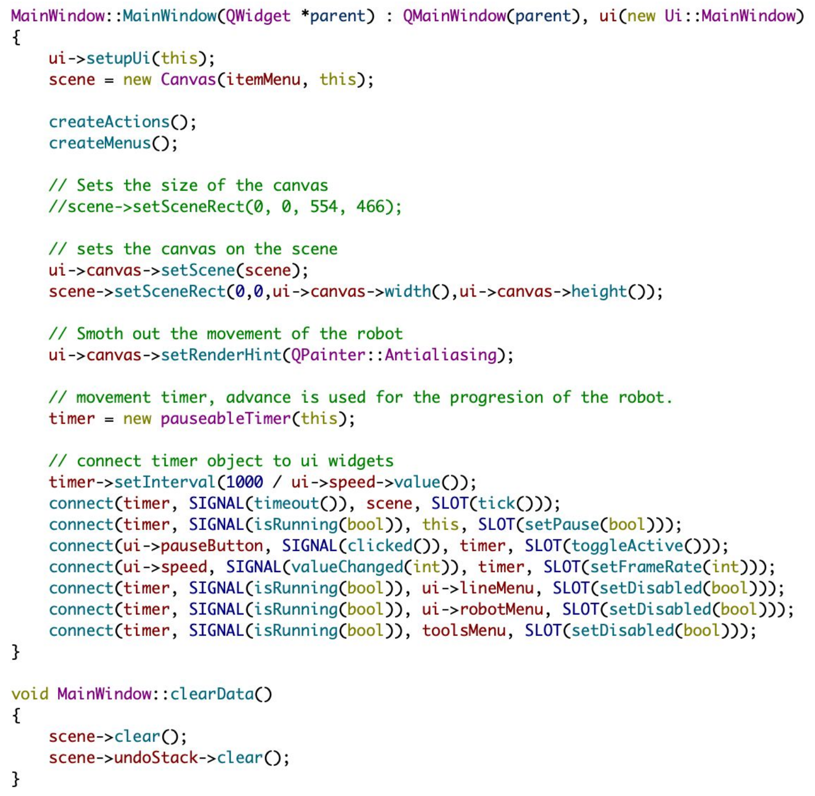
Line Robots utilizes the single responsibility principle by having each class have a single purpose with all methods relating to the function. When the program was initially designed the goal was to have each class to have a single responsibility and thus only a single reason to change.

In addition, Line Robots utilizes the open/closed principle by having all aspects of architecture such as classes, modules, functions, to be open for extension at any time. At the same time all the components of the Line Robots architecture are closed for modification.

Lastly, Line Robots utilizes the interface segregation principle where users and clients do not have to be forced to depend upon interfaces that they do not use. Rather, they have the ability to choose what they need, and the program will continue to function.

In terms of design Line Robots uses object oriented programming (OPP) that revolves around using objects and declaration of classes to achieve a simple and reusable design. OPP was used to increase the longevity of the program and to increase the maintainability in case the program needs to be updated to a newer software standart or due to decay. In the Figure 1B a general class diagram is shown of all the class components of the program. The main function is responsible for initiating all the class elements in the program. As shown in the diagram, the main function only contains the necessary function required to initiate the front end and back end of the program. Each class in the program is independent and can be programed or updated individually without breaking functionality.

The main function calls the mainwindow front end GUI and mainwindow class. Mainwindow class is responsible for leading all the GUI elements in the front end and sets up the back end coded in C++ further. From the mainwindow class the program is connected to all other classes via signals and slots. Every time a user clicks on an element on the GUI the signal is sent to the code. Connecting a button's clicked() signal to a single method that responds appropriately is probably the most common connection scenario.



```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    scene = new Canvas(itemMenu, this);

    createActions();
    createMenus();

    // Sets the size of the canvas
    //scene->setSceneRect(0, 0, 554, 466);

    // sets the canvas on the scene
    ui->canvas->setScene(scene);
    scene->setSceneRect(0,0,ui->canvas->width(),ui->canvas->height());

    // Smoth out the movement of the robot
    ui->canvas->setRenderHint(QPainter::Antialiasing);

    // movement timer, advance is used for the progresion of the robot.
    timer = new pauseableTimer(this);

    // connect timer object to ui widgets
    timer->setInterval(1000 / ui->speed->value());
    connect(timer, SIGNAL(timeout()), scene, SLOT(tick()));
    connect(timer, SIGNAL(isRunning(bool)), this, SLOT(setPause(bool)));
    connect(ui->pauseButton, SIGNAL(clicked()), timer, SLOT(toggleActive()));
    connect(ui->speed, SIGNAL(valueChanged(int)), timer, SLOT(setFrameRate(int)));
    connect(timer, SIGNAL(isRunning(bool)), ui->lineMenu, SLOT(setDisabled(bool)));
    connect(timer, SIGNAL(isRunning(bool)), ui->robotMenu, SLOT(setDisabled(bool)));
    connect(timer, SIGNAL(isRunning(bool)), toolsMenu, SLOT(setDisabled(bool)));
}

void MainWindow::clearData()
{
    scene->clear();
    scene->undoStack->clear();
}
```

The picture on the left shows a sample code of the MainWindow class found in the program. The functions such as pauseButton are connected via SIGNALS and SLOTS using the connect() function. Each signal is then passed into each own separate class. For the ability for a user to pause the timer the signal that is received is passed into pauseabletimer class where the program receives it in the back end and initiates the response to pause the simulation.

Running the Program

Sequence Diagram (Figure 1C) shows the usage scenarios of how Line Robot program is intended to work. The sequence diagram allows for a visual representation of the logic that was implemented by Team Bionics of every usage scenario. This sequence diagram was refixed countless times until it was perfected in both usability and logic. For example, a robot is placed on one the line, and then immediately the user decides to add in three more robots. Therefore, a pause button was necessary to be implemented to refine the use case. Furthermore, a setSpeed function was implemented to increase the usability of the program allowing for a further exploration of logic of methods. Thus, the sequence diagram is a visual representation of the object code.

Class Diagram for Line Robots

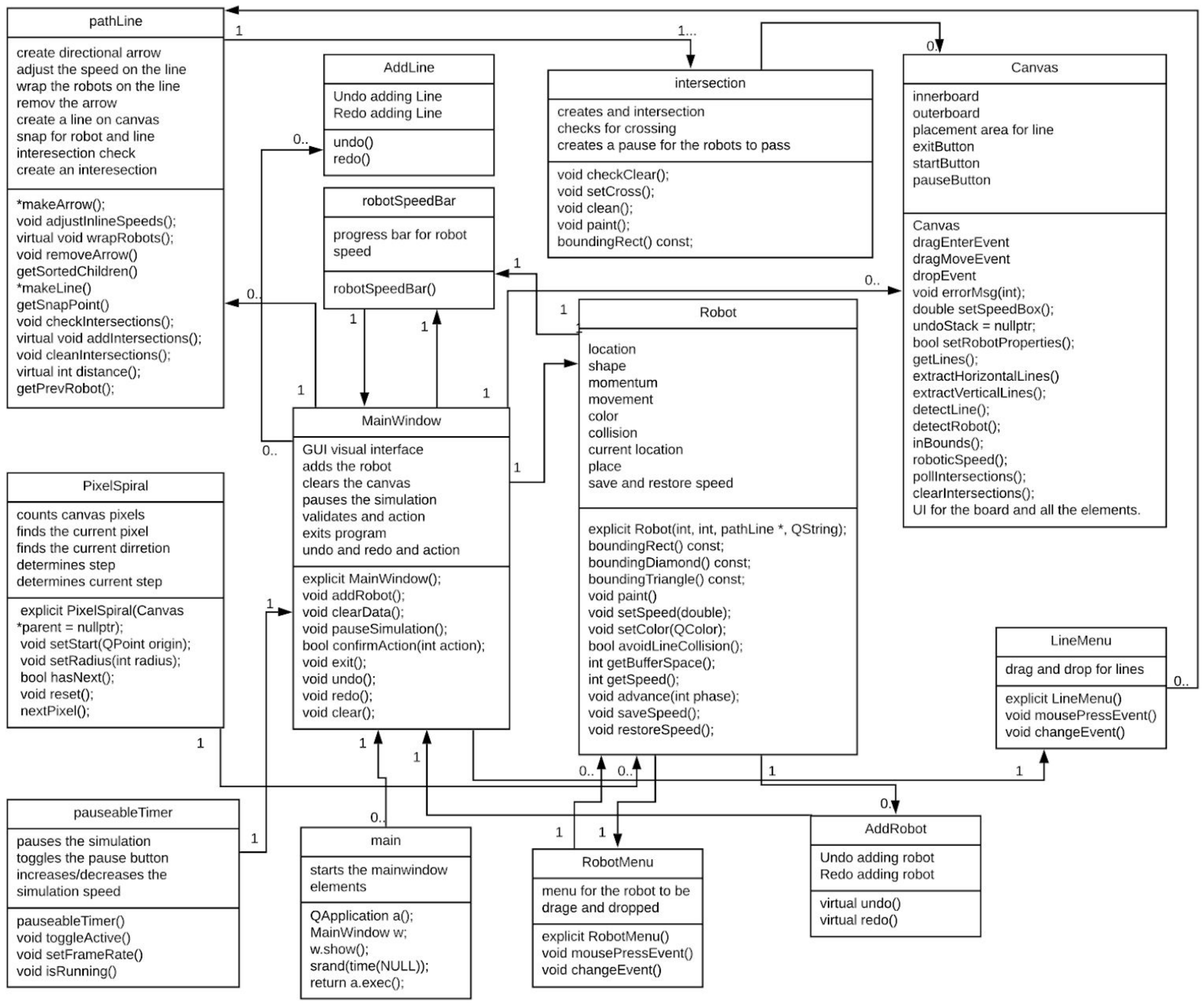
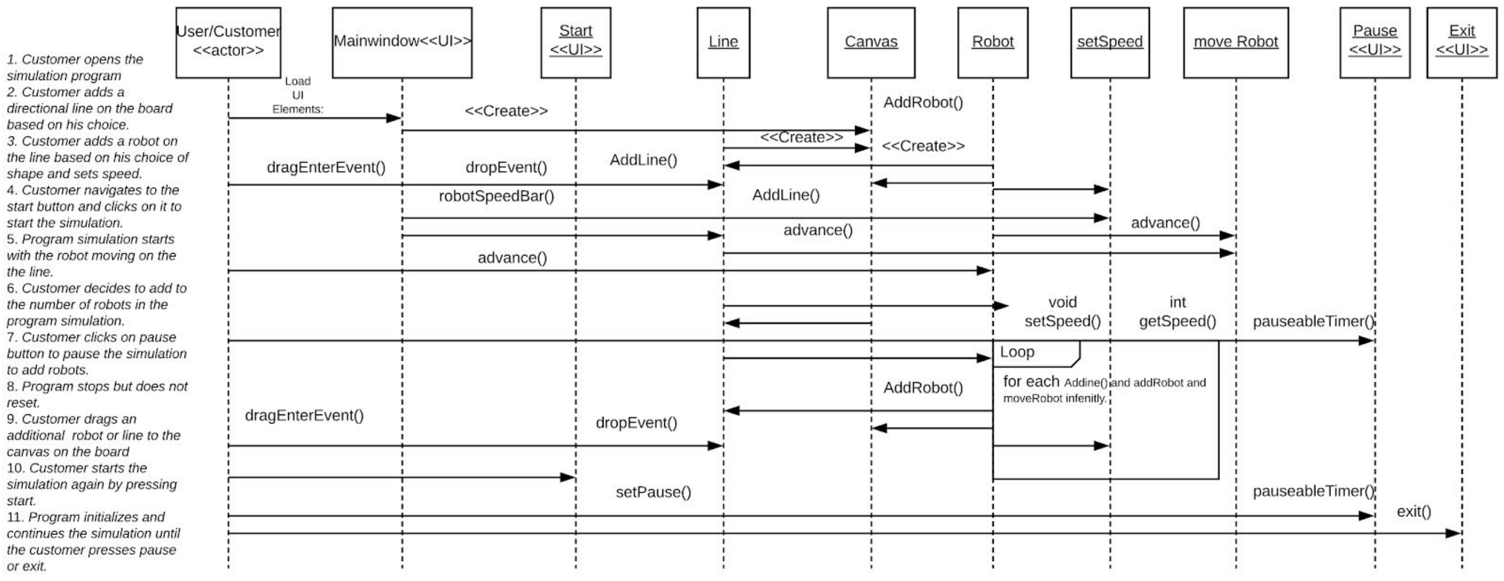


Figure 1B

General Class Diagram of Line Robots

Sequence Diagram for Line Robots



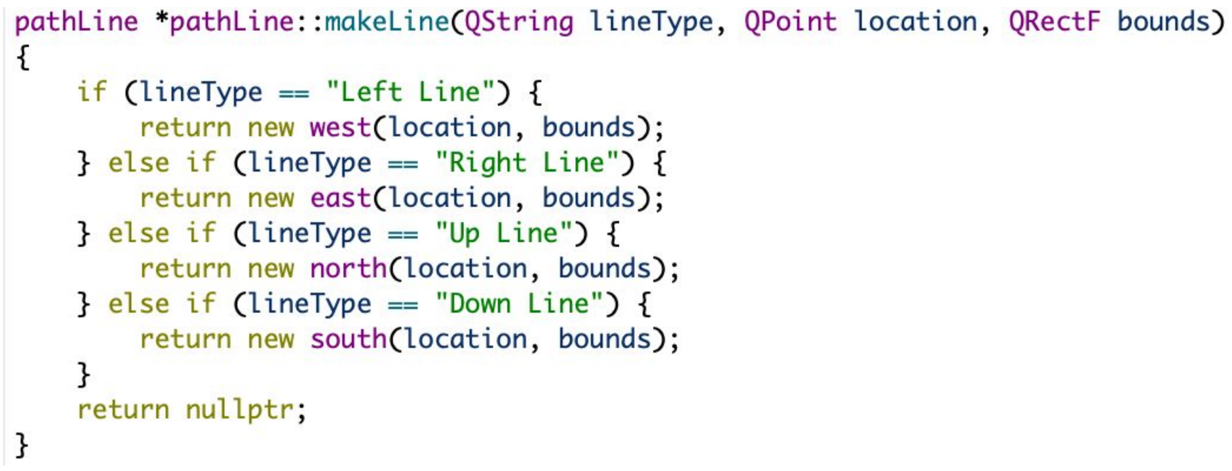
1. Customer opens the simulation program
2. Customer adds a directional line on the board based on his choice.
3. Customer adds a robot on the line based on his choice of shape and sets speed.
4. Customer navigates to the start button and clicks on it to start the simulation.
5. Program simulation starts with the robot moving on the line.
6. Customer decides to add to the number of robots in the program simulation.
7. Customer clicks on pause button to pause the simulation to add robots.
8. Program stops but does not reset.
9. Customer drags an additional robot or line to the canvas on the board
10. Customer starts the simulation again by pressing start.
11. Program initializes and continues the simulation until the customer presses pause or exit.

Figure 1C
General Sequence Diagram of Line Robots

1. Customer opens the simulation program
2. Customer adds a directional line on the board based on his choice.
3. Customer adds a robot on the line based on his choice of shape and sets speed.
4. Customer navigates to the start button and clicks on it to start the simulation.
5. Program simulation starts with the robot moving on the line.
6. Customer decides to add to the number of robots in the program simulation.
7. Customer clicks on the pause button to pause the simulation to add robots.
8. Program stops but does not reset.
9. Customer drags an additional robot or line to the canvas on the board
10. Customer starts the simulation again by pressing start.
11. Program initializes and continues the simulation until the customer presses pause or exit.

Line Robots design pattern(s) and in which modules/components is used.

The line_robots project uses a factory pattern to create an appropriate concrete object whenever a caller requests that a pathLine be created with a certain direction. The north, south, east, and west objects are all derived from pathLine, but have different behaviors for their child robots and for sorting and wrapping based on their direction. A calling function assigns the return value of the makeLine() function to a pathLine pointer. The makeLine factory method determines which concrete pathLine-derived class is needed based on a parameter and returns a pointer to a new concrete pathLine-derived object.
In the context of line_robots, this is used by the Canvas object to create a pathLine in response to a drop event. Each drop of a line on the canvas triggers a call to makeLine, with a parameter indicating the concrete type of line to be created.



```
pathLine *pathLine::makeLine(QString lineType, QPoint location, QRectF bounds)
{
    if (lineType == "Left Line") {
        return new west(location, bounds);
    } else if (lineType == "Right Line") {
        return new east(location, bounds);
    } else if (lineType == "Up Line") {
        return new north(location, bounds);
    } else if (lineType == "Down Line") {
        return new south(location, bounds);
    }
    return nullptr;
}
```

Figure 2B

Sample Code showing the implementation of the factory pattern

The final state/condition of the Line Robots

Current Status of Line Robots

As of the current build (v0.03), the software is completed. The program functions based on the design and requirements stated by the customer.

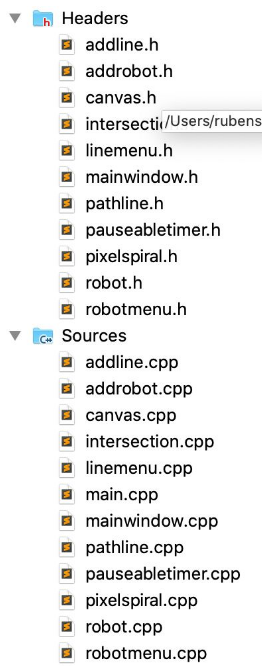
Bugs and Issues

1. Robots occasionally experience near-collisions, since the internal representation of the robot is a point, but the graphical representation has a 2-D size.
2. Intersections only detect approaching robots, so another near-collision scenario can occur where a fast robot passes close behind a slow robot that has just passed an intersection.

Product Backlog: To be implemented

Currently there are no more stories left in the product backlog. All the stories that were in the backlog were implemented.

A explanation of the general file structure of the Line Robots



- Headers
  - addline.h
  - addrobot.h
  - canvas.h
  - intersecti /Users/rubens
  - linemenu.h
  - mainwindow.h
  - pathline.h
  - pauseabletimer.h
  - pixelspiral.h
  - robot.h
  - robotmenu.h
- Sources
  - addline.cpp
  - addrobot.cpp
  - canvas.cpp
  - intersection.cpp
  - linemenu.cpp
  - main.cpp
  - mainwindow.cpp
  - pathline.cpp
  - pauseabletimer.cpp
  - pixelspiral.cpp
  - robot.cpp
  - robotmenu.cpp

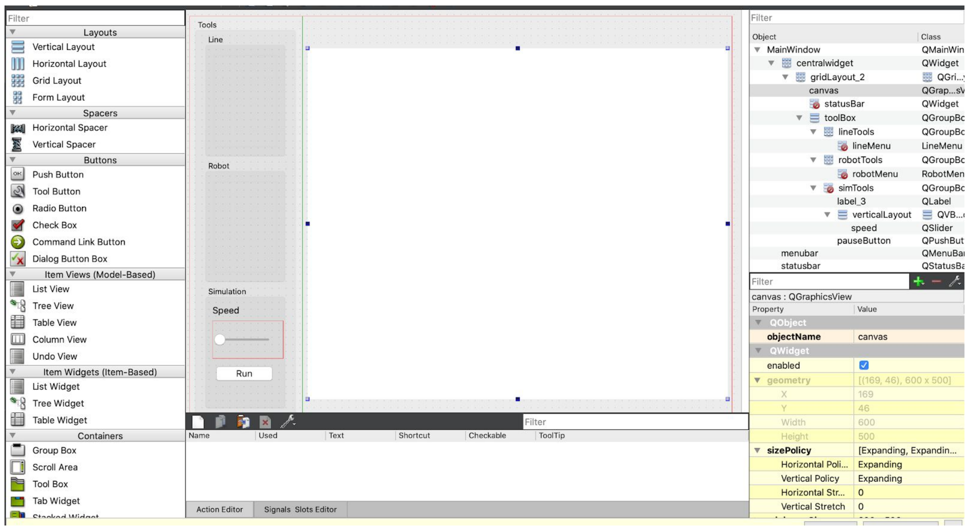
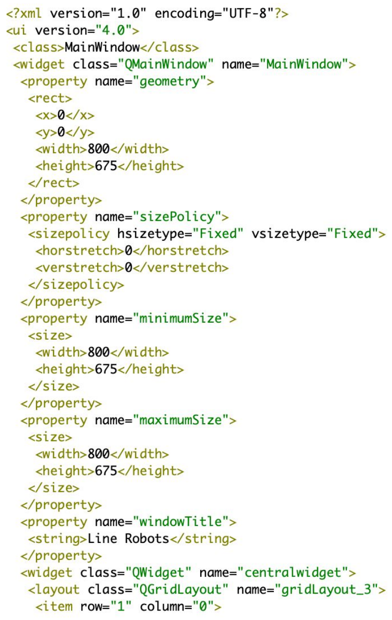


Figure 3a
(General view of the files)

Figure 3B
(General view of the GUI edit)

Line Robots utilizes object oriented programming. Each feature of the software uses classes which are linked together to produce the final program. At the core of the back end, Line Robots uses C++. While at the backend Qt is used to provide a visual representation of the code in terms of a graphical user interface (GUI).

General Overview



```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>675</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="minimumSize">
   <size>
    <width>800</width>
    <height>675</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>800</width>
    <height>675</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Line Robots</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout_3">
    <item row="1" column="0">
```

The program at its core is composed of four main file categories. The first file categorie is the header and program files. These files act as modules for the software and are the backbone one the program. These files are implemented and run in the backend to promote all the features in the GUI (Figure 3a).

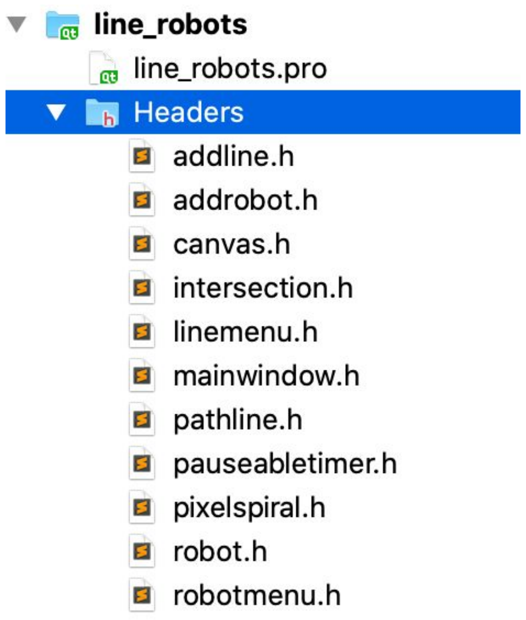
Second file categorie are the “Form”, which contain the GUI elements written in a file called “mainwindow.ui”. The mainwindow.ui contains the code for all the “widgets” that the users see when they run the program. As at is shown on the left. The GUI builder (Figure 3B) allows for precise adjustments of all UI elements.

The GUI utilizes signals and slots promoted from the backend of the C++ code. See figure 3B for an overview of the GUI interface system. Third file categorie are the “Resource” files. These are files that the program uses to import them into the GUI. In the case of Line Robots these are images of shapes for the robots and the directional lines. The program imports these images into the GUI and uses them as elements as if they were native to the software.

Class File Structure

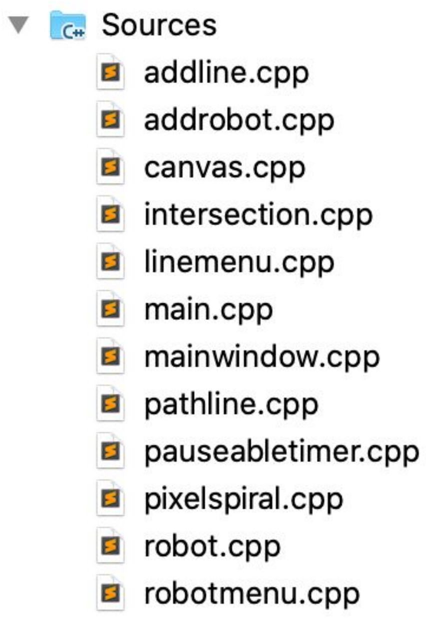
- AddLine: An AddLine is an extension of the QUndoCommand class provided by the Qt framework, which encapsulates the code required to add a line to the drawing canvas via the drag/drop interface so that this action can be stored in an undo stack. This class enables the undo/redo user interface for lines.
- AddRobot: An AddRobot is an extension of the QUndoCommand class provided by the Qt framework, which encapsulates the code required to add a robot to the drawing canvas via the drag/drop interface so that this action can be stored in an undo stack. This class enables the undo/redo user interface for robots.
- Canvas: A Canvas is an extension of the QGraphicsScene class provided by the Qt framework. The Canvas class serves as a container for the robot and line objects the user creates, and acquires these children by implementing the drop portion of the drag and drop interface. The canvas also orchestrates the movement of the robots by signaling them to advance when the Canvas’s tick() slot it triggered. For each tick(), the Canvas has each pathLine check all intersections and set up traffic signals, tells all Robots to move 1 frame, and clears the traffic signals. Each call to tick() can be considered 1 “frame” of the simulation.

- Intersection: An intersection is an extension of the QGraphicsItem class provided by the Qt framework. When a pathLine is created, it checks of it crosses any existing pathLines. At each of these crossings, an intersection object child is created that also links to the other pathLine. An intersection object can, given a pointer to the nearest approaching Robot on its parent pathLine, determine whether this robot should have priority or the closest approaching robot on the crossing pathLine should have priority. The appropriate pathline is then "blocked" by placing an invisible 0-speed robot on the pathLine to be blocked.
- LineMenu: A LineMenu is a container that holds representations of lines that the user can drag onto the canvas to draw paths. This implements the drop portion of the drag/drop interface.
- RobotMenu: A RobotMenu is a container that holds representations of lines that the user can drag onto a line in the canvas to draw robots. This implements the drop portion of the drag/drop interface.
- MainWindow: MainWindow creates the graphical user interface window based on the form created in the WYSIWYG interface editor. MainWindow also performs housekeeping tasks like connecting signals to their requisite slots to support operation of the program.
- pathLine: The pathLine family of objects implements extensions to the QGraphicsLineItem class provided by the Qt framework. A pathLine is generated using a factory pattern as a north, a south, an east, or a west concrete representation. A pathLine serves as the parent for the robots placed on it, as well as the intersections it generates on creation. The pathLine handles wrapping Robots back to the beginning of the line when they reach the end. A pathLine also supports determining the Robot closest approaching to a point on itself. The pathline.h and pathline.cpp files include the code for the north, south, west, and east concrete classes.
- pausableTimer: The pausable timer wraps the QTimer class provided by the Qt framework with slots to support being able to pause and restart the pulses generated. In the context of line_robots, the pausableTimer is used to trigger the simulation to advance 1 frame each time the timer hits its count.
- PixelSpiral: A PixelSpiral generates a series of coordinates in its parent QGraphicsScene that expand outward in a clockwise spiral from an origin point. In the context of line_robots, this class is used to locate the closest line to where a robot is dropped so that a Robot can be created and snapped to that pathLine.
- Robot: A Robot extends the QGraphicsItem class provided by the Qt framework to represent a collision avoiding robot. Upon request, given the distance to and object ahead and a pointer to that object, the robot determines whether it needs to slow down to avoid a collision. If the path ahead is clear, the robot accelerates towards its maximum speed. In the context of line_robots, these behaviors are triggered by the Canvas object, which sends Qt advance() signals to all of its descendants. The advance() signal in Qt has 2 phases, a prep, or 0 phase, and a move, or 1 phase. On the 0-phase, pathLines trigger their Robot children to adjust their speeds, then, on the 1-phase, the Robots move based on their currently set speed.



line_robots
line_robots.pro
Headers
addline.h
addrobot.h
canvas.h
intersection.h
linemenu.h
mainwindow.h
pathline.h
pauseabletimer.h
pixelspiral.h
robot.h
robotmenu.h

Figure 4A



Sources
addline.cpp
addrobot.cpp
canvas.cpp
intersection.cpp
linemenu.cpp
main.cpp
mainwindow.cpp
pathline.cpp
pauseabletimer.cpp
pixelspiral.cpp
robot.cpp
robotmenu.cpp

Figure 4B

Figure 4 shows the General File Structure Described Above

Future Development and Contribution of Line Robots

Team Bionics decided that the keeper of the project will bet. Team Bionics decided that for future contributions on this project Alex Maslar (github: https://github.com/alexmaslar) To become a contributor to Line Robots the developer will need to contact Alex. The developer who wants to further contribute to the project needs to know the current bugs and read the documentation.