Get Started With StarL

Yixiao Lin

lin187@illinois.edu

This is a detailed guide to get started with StarL. We will first give you detailed instructions on how to set up StarL. Then, we will show how to run examples and briefly explain them.

Setting up StarL:

StarL is based on java platform with some Android extensions. Therefore, you need to use ADT(Android developing tool) or the Android Studio to run StarL. JRE(java running environment) and JDK(java development kit) are needed to run ADT or Android Studio.

Download and install the JDK1.7 and JRE1.7 at

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Download and install Android Studio or ADT:

http://developer.android.com/sdk/index.html

https://developer.android.com/sdk/installing/index.html?pkg=studio

In the welcome page of Android Studio, click Configure, SDK Manager.

In eclipse, click Window, Android SDK Manager.

Install the latest SDK(API21) using the SDK manager, if you are having problems, try install SDK 2.3.3(API 10). If you are using Android Studio, set the language level to 7.0 Diamonds, ARM, multi-catch etc.

https://developer.android.com/sdk/installing/adding-packages.html

After this step, your Android Studio or ADT should be ready to import StarL project.

In addition, you can download and install git for source control:

http://git-scm.com/

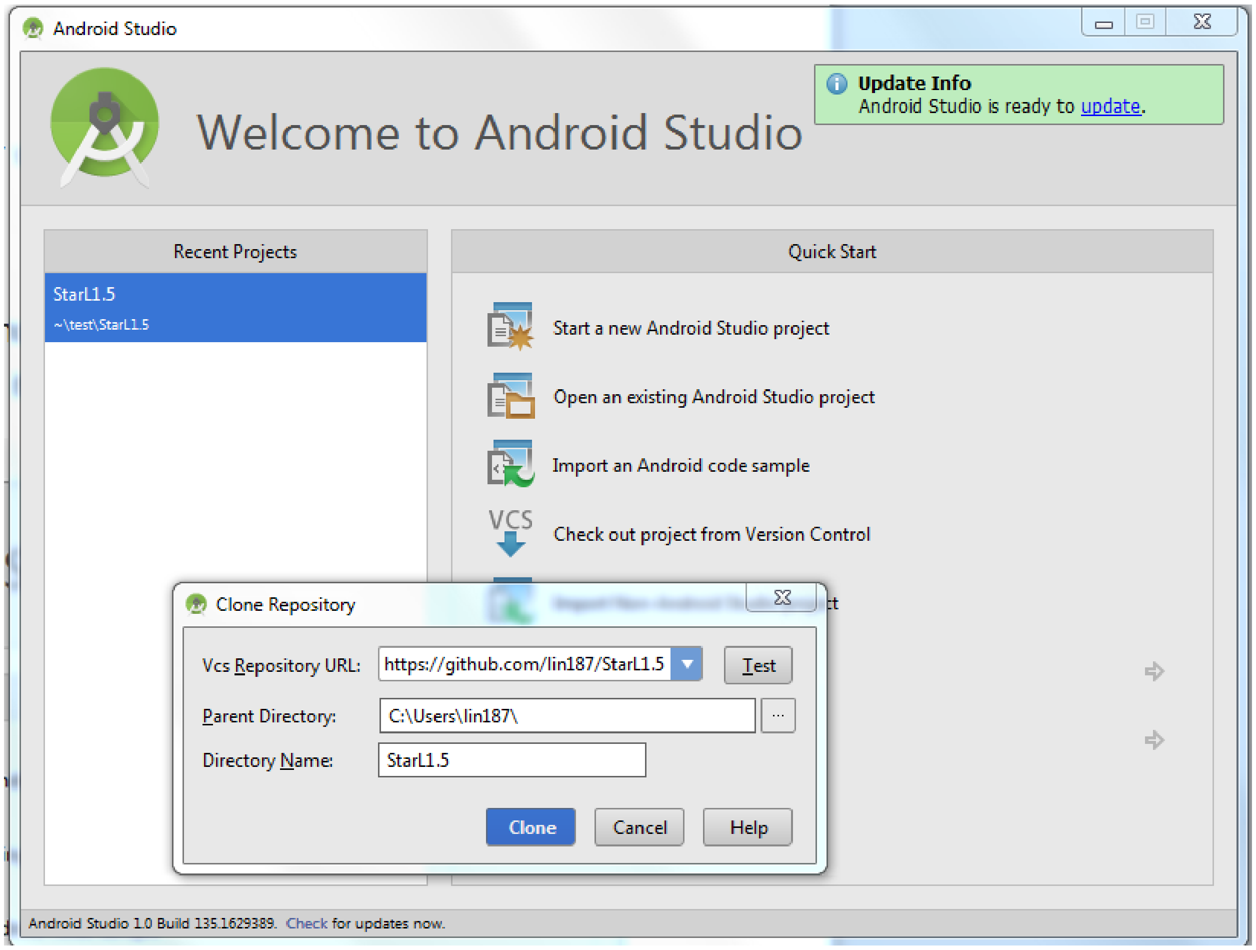
Now, get the StarL1.5 source code from github:

https://github.com/lin187/StarL1.5/

You can download it as zip, clone it using git or clone it using Android Studio.

In Android Studio:

Click check out project from Version Control, choose git, and enter the StarL1.5 link. Choose a path to store this repository.


Android Studio
Welcome to Android Studio
Update Info
Android Studio is ready to update.
Recent Projects
StarL1.5
~\test\StarL1.5
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Import an Android code sample
Check out project from Version Control
Clone Repository
Test
Clone
Cancel
Help
Android Studio 1.0 Build 135.1629389. Check for updates now.

Vcs Repository URL: https://github.com/lin187/StarL1.5
Parent Directory: C:\Users\lin187\
Directory Name: StarL1.5

This will load the StarL into this directory and import it into Android Studio.

If you download it as zip or cloned it using git to a directory named *myStarLDir,* you can import it.

In Android Studio welcome page, click Open an existing Android Studio project, choose the directory your download StarL: *myStarLDir.* A number of items will appear in the Projects list. Select All. Click OK.

In eclipse, click file, import, General, Existing Projects into Workspace. Choose the directory your download StarL: *myStarLDir.* A number of items will appear in the Projects list. Select All. Click finish.

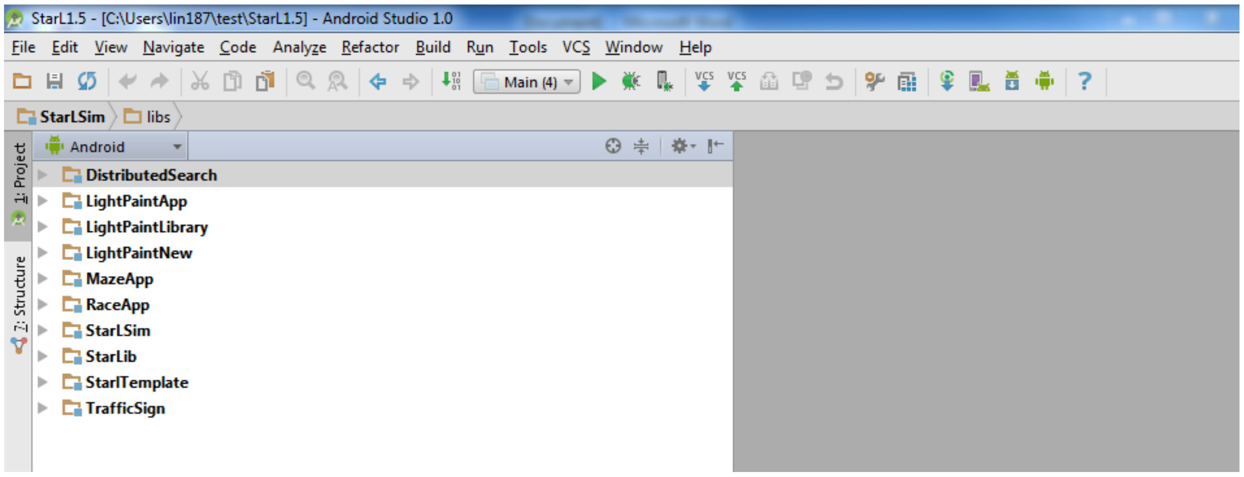
The starlLib and starlSim are core of StarL.

RaceApp, MazeApp, TrafficSignApp, DistributedSearchApp, LightPaintingSim are StarL simulation applications.

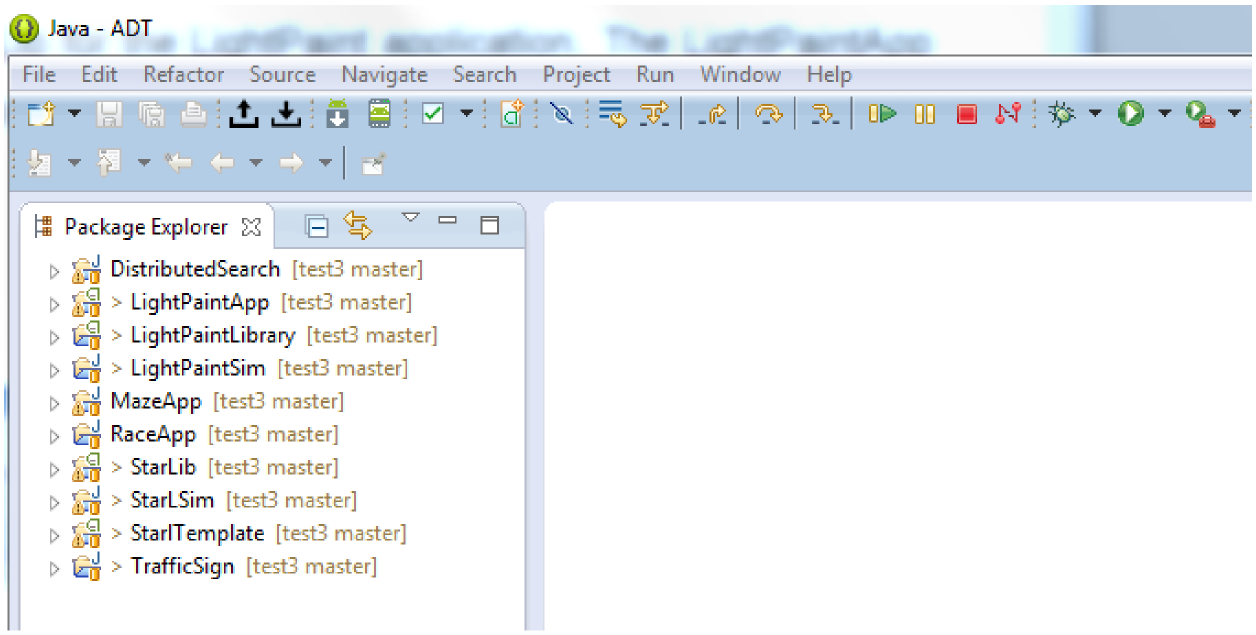
LightPaintLib is extension to StarLib for the LightPaint application. The LightPaintApp and StarLtemplete are android applications.

Wait until everything is load up.

You should see things like this in Android Studio:



In eclipse:

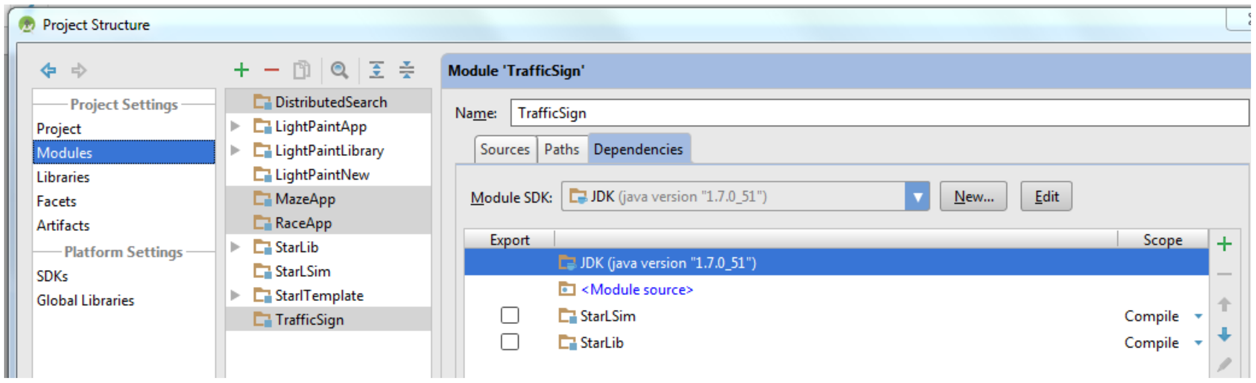

Java - ADT
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer

DistributedSearch [test3 master]
> LightPaintApp [test3 master]
> LightPaintLibrary [test3 master]
> LightPaintSim [test3 master]
MazeApp [test3 master]
RaceApp [test3 master]
> StarLib [test3 master]
> StarLSim [test3 master]
> StarlTemplate [test3 master]
> TrafficSign [test3 master]

In Android Studio, click File, Project Structure. Set Project language level to 7.0 if it is not already set.

You should be able to run StarL examples. If compile error remains, try setting the project dependencies like the following:

DistributedSearch, MazeApp, RaceApp, TrafficSign should have same dependencies: JDK 1.7, StarLib, StarLSim and their own source files.



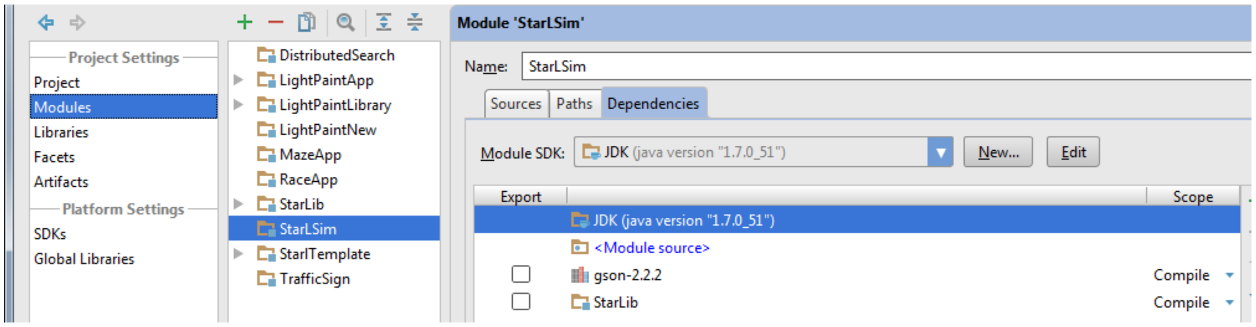
Module 'TrafficSign'

Name: TrafficSign

Module SDK: JDK (java version "1.7.0_51")

Export		Scope
	JDK (java version "1.7.0_51")	
	<Module source>	
☐	StarLSim	Compile
☐	StarLib	Compile

StarLSim



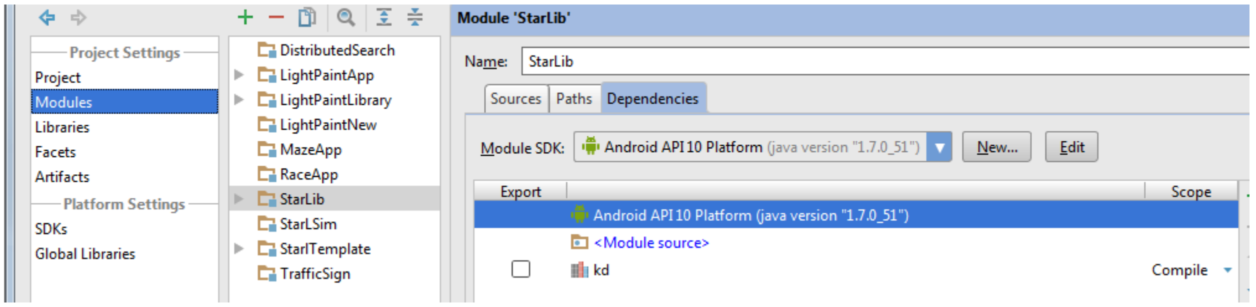
Module 'StarLSim'

Name: StarLSim

Module SDK: JDK (java version "1.7.0_51")

Export		Scope
	JDK (java version "1.7.0_51")	
	<Module source>	
☐	gson-2.2.2	Compile
☐	StarLib	Compile

StarLib



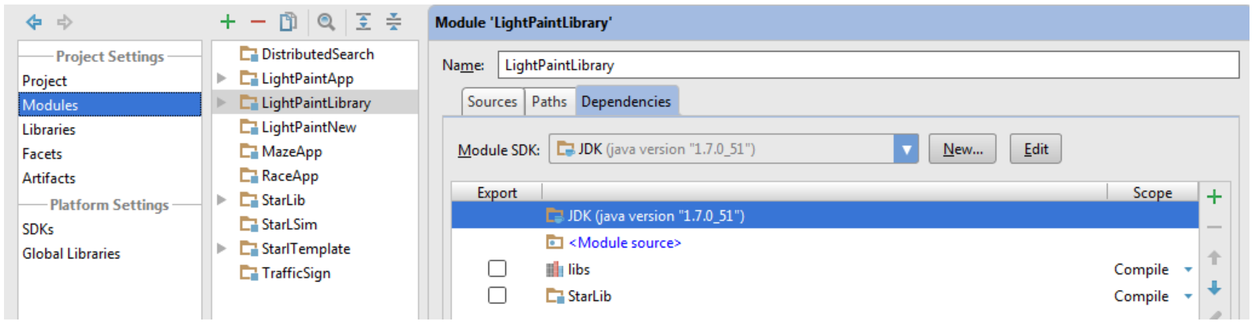
Module 'StarLib'

Name: StarLib

Module SDK: Android API 10 Platform (java version "1.7.0_51")

Export		Scope
	Android API 10 Platform (java version "1.7.0_51")	
	<Module source>	
☐	kd	Compile

LightPaintLibrary



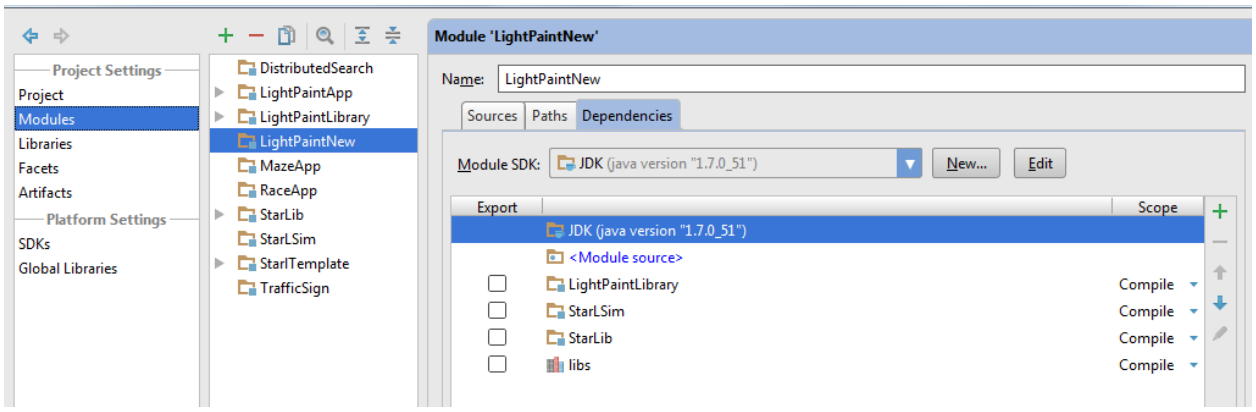
Module 'LightPaintLibrary'

Name: LightPaintLibrary

Module SDK: JDK (java version "1.7.0_51")

Export		Scope
	JDK (java version "1.7.0_51")	
	<Module source>	
☐	libs	Compile
☐	StarLib	Compile

LightPaintNew



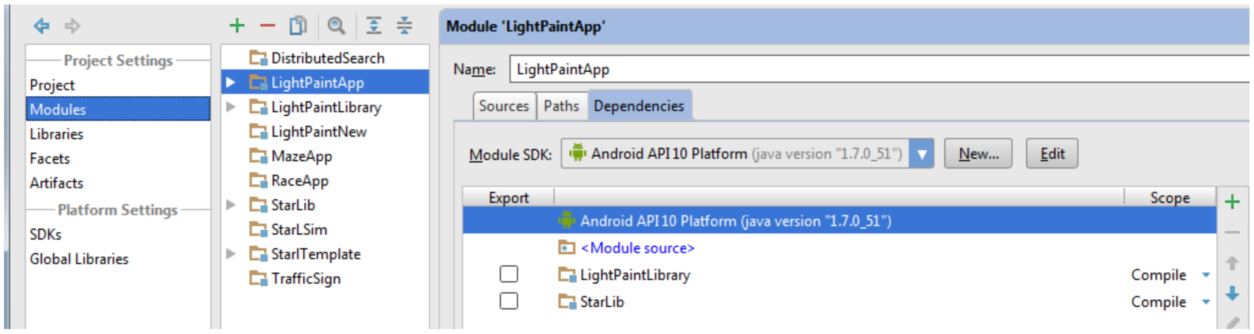
Module 'LightPaintNew'

Name: LightPaintNew

Module SDK: JDK (java version "1.7.0_51")

Export		Scope
	JDK (java version "1.7.0_51")	
	<Module source>	
☐	LightPaintLibrary	Compile
☐	StarLSim	Compile
☐	StarLib	Compile
☐	libs	Compile

LightPaintApp



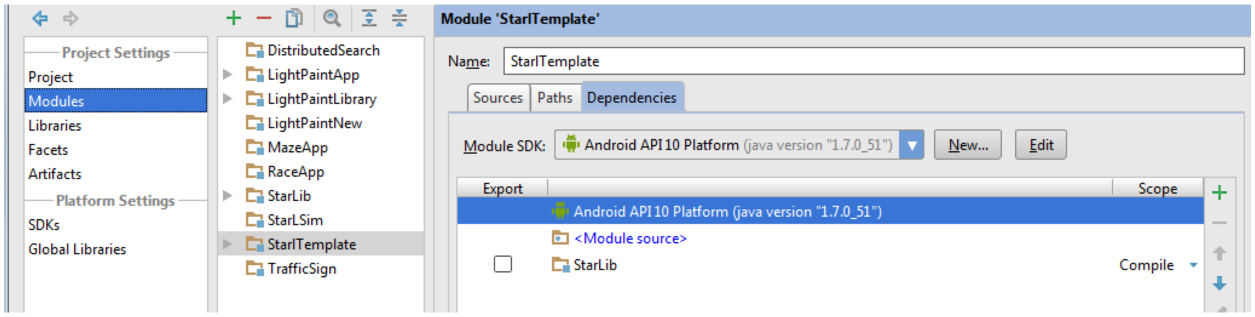
Module 'LightPaintApp'

Name: LightPaintApp

Module SDK: Android API 10 Platform (java version "1.7.0_51")

Export		Scope
	Android API 10 Platform (java version "1.7.0_51")	
	<Module source>	
☐	LightPaintLibrary	Compile
☐	StarLib	Compile

StarlTemplate



Module 'StarlTemplate'

Name: StarlTemplate

Module SDK: Android API 10 Platform (java version "1.7.0_51")

Export		Scope
	Android API 10 Platform (java version "1.7.0_51")	
	<Module source>	
☐	StarLib	Compile

In eclipse, right click each project, click properties, under java build path, set similarly under Projects, Libraries. Also, in each project Properties dialog in the Java Compiler section, ensure that the Enable project specific settings checkbox is unchecked.

If error remains, try removing LightPaintApp and StarlTemplate, you do not need these two to run the examples.

If you cannot resolve compiler issues, feel free to contact us. We will try our best to assist you.

StarL Examples:

The examples are RaceApp, MazeApp, TrafficSign, LightPaintNew and DistributedSearch.

Each of them includes three java source files, Main.java, someApp.java, someDrawer.java. Take RaceApp as our example:

Main file set some values for simulation, for example, number of robots, how fast the time goes. It also creates the simulation application and launches it.

MazeApp defines the state machine that controls the robot behavior.

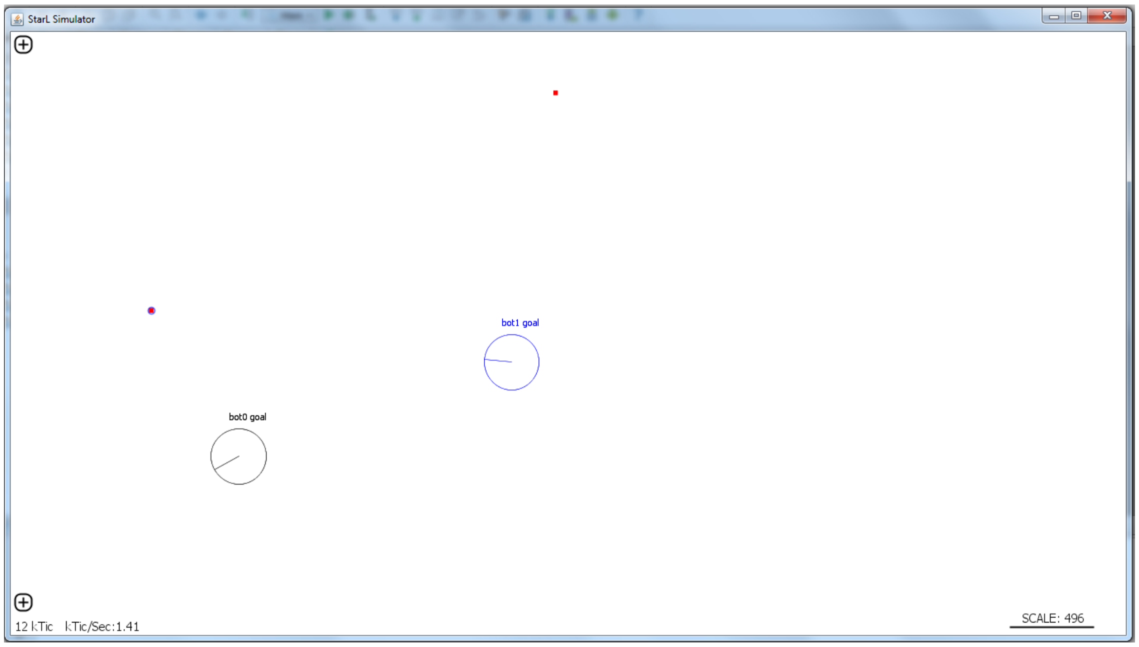
MazeDrawer draws extra information on the 2D visualizer. We will see that when simulating StarL.

To run RaceApp simulation:

In Android Studio: right click Main.java under RaceApp source folder. Run Main.main()

In Eclipse: right click Main.java under RaceApp source folder. Run as, Java Application

You should see something like this:

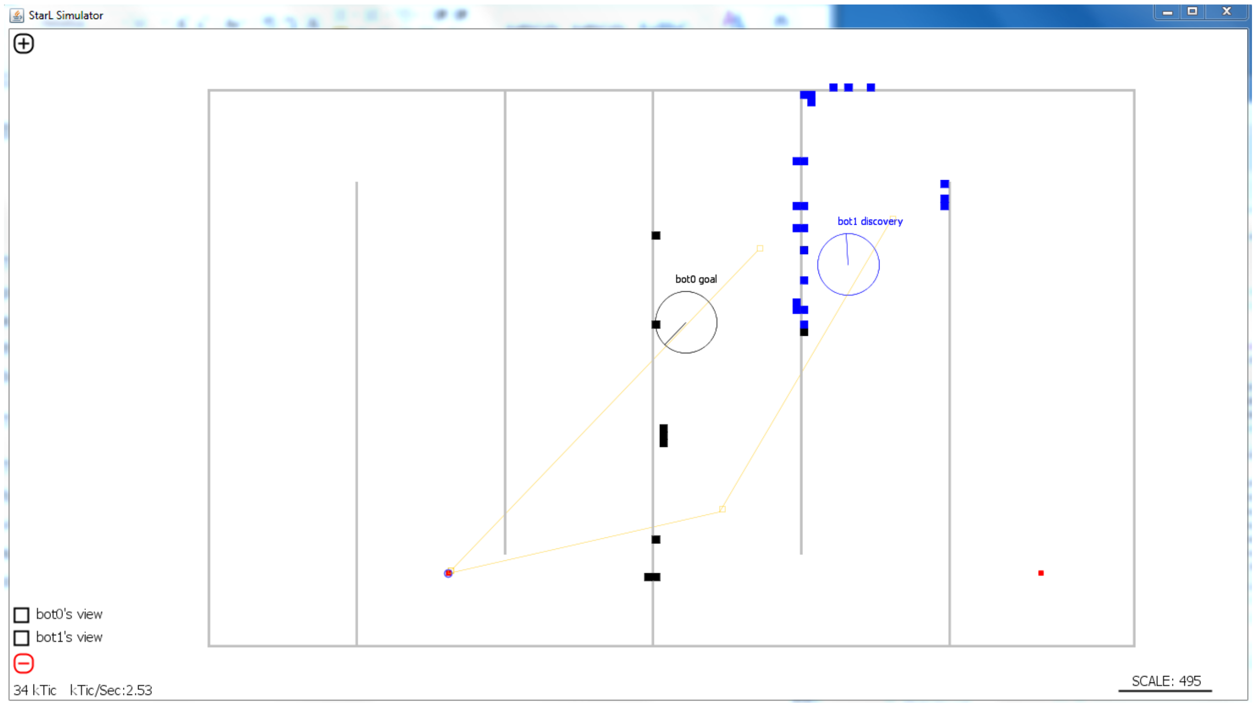


The RaceApp is a simulation where a number of robots (iRobot Create) starting in any point in the map. Their goal is to reach a number of destinations. Once a robot reaches a destination, it announces to other robots. Then they will race to the next point. When the robots collide, they will stop and start turning. They will stop when all points have been reached.

We suggest you use RaceApp as a template when you develop a new simulation application using StarL.

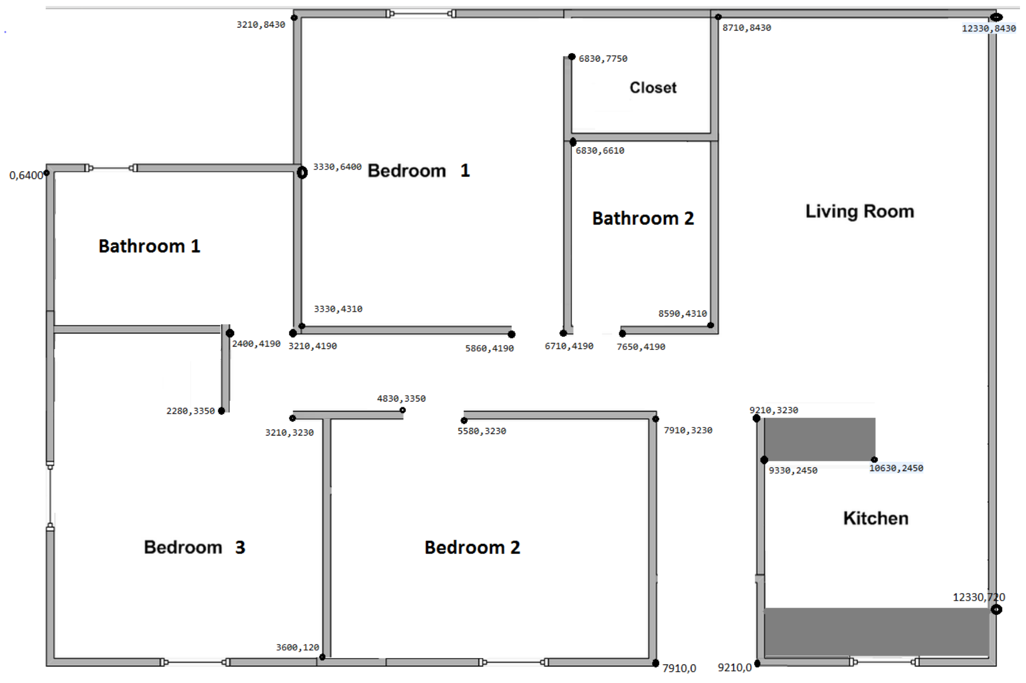
The MazeApp is a simulation for path planning in a maze like environment. Every robot will try to reach a point at the bottom right corner. They will use RRT path planning to avoid colliding with walls. However, initially, they do not know how the maze looks like. They will learn about the position of walls when the bump into the wall. Different types of robots are included in the simulation to demonstrate their behavior. There are no robot coordination written in this app yet, you can add them if you want them to be more efficient.

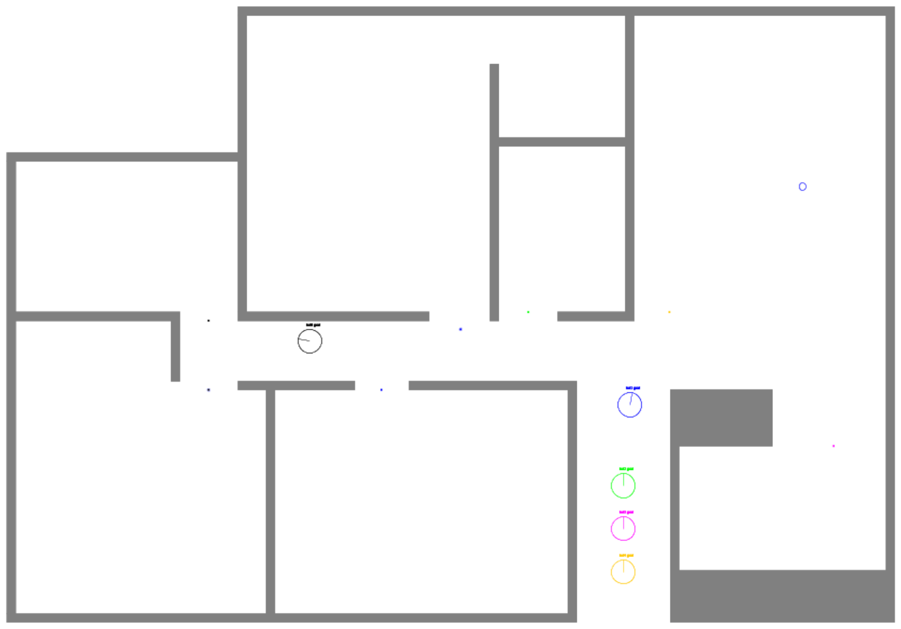
After the simulation start, click the plus sign at the bottom left corner, uncheck box bot0's view. You will see how bot0 see the environment now. Different robot's view are shown in different color.



The DistributedSearch App simulates robots searching for objects in a room. They will first elect a leader, then the leader will assign each robot some rooms to cover. The robots will go ahead to cover the rooms. If the object is close to the robot without a wall in between, they the object is found (sensed). When the object is found, announce it to other robots.

The room and one simulation:





There are some resource files for each StarL App; we explain them here using DistributedSearch as an example. Under DistributedSearch/waypoints folder (resource folder for Android Studio), dest.wpt specifies the points for the robots to visit; Obstacles.wpt specifies the location of obstacles in the environment; start.wpt specifies where each robot initially start, senseObjects.wpt specifies where the sensible objects are in the map.

Dest.wpt has format of: WAY,x,y,angle,name,radius

Obstacles.wpt has format of: Obstacle;x1,y1;x2,y2;x3,y3;x4,y4;time

The four points are four edge points for a rectangle. Time is for how long this obstacle will last, -1 means it lasts forever.

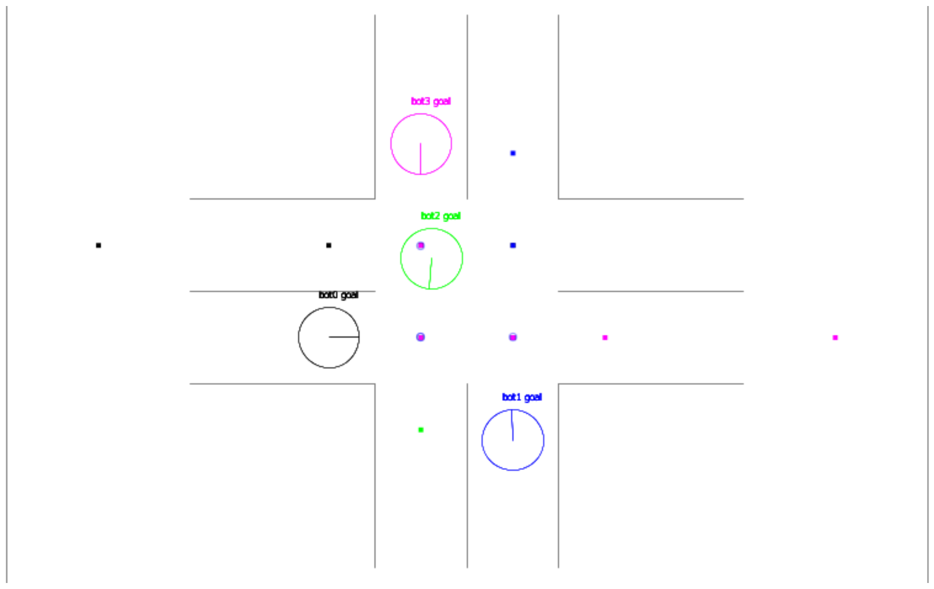
Start.wpt has format of: WAY,x,y,angle,robotName

Sense.wpt has format of: SENSE,x,y,angle,Name

You can modify some points in these files to see how simulation differs.

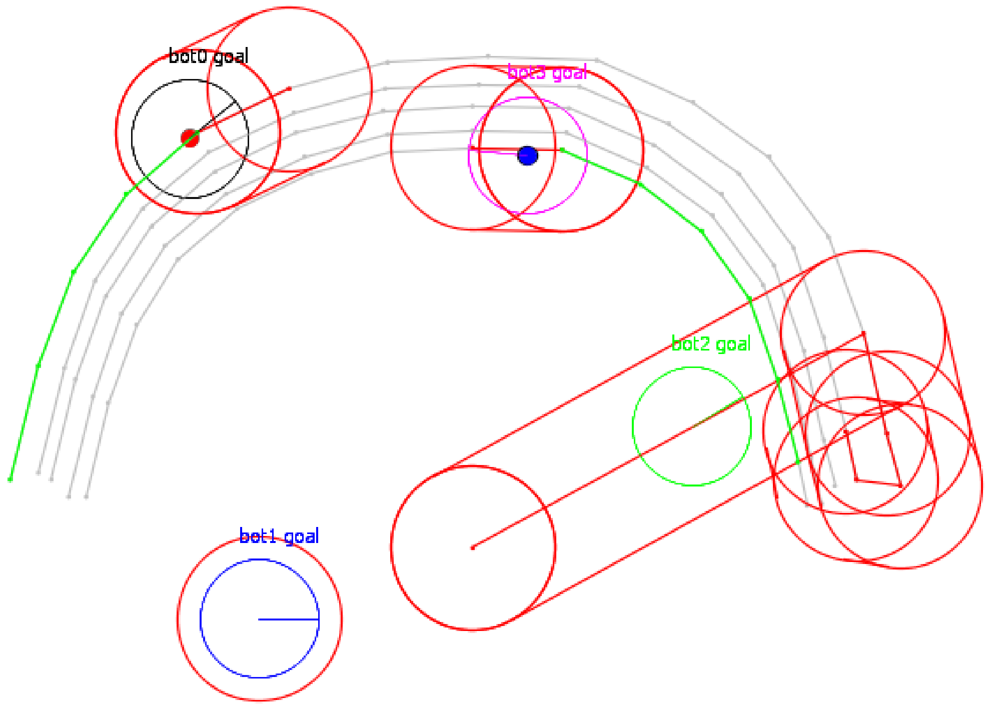
The TrafficSign App simulates a four way stop sign intersection for robots. We implemented a mutual exclusion algorithm so that the robots go through the intersection safely and efficiently. See StarLTrafficSignApp.pdf for additional material on the TrafficSign App.

One simulation:



The LightPaint App simulates robot light paint a picture. Given a picture, the robots will try cover the image and display different color on different segments. See https://www.youtube.com/watch?v=QNr8h9c8nBM

One Simulation:


bot0 goal
bot3 goal
bot2 goal
bot1 goal