Getting 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.

To being using StarL, you will need to install Android Studio or Eclipse with ADT.

Setting up StarL with Android Studio (preferred):

Android Studio is the official IDE for developing Android applications. To install Android Studio, follow the instructions on the webpage linked to below. Note that you will need to install JDK version 7 before installing Android Studio if you haven't already. A link to download JDK is provided on the webpage below.

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

After installing, start Android Studio and a Setup Wizard will appear. Click configure, then SDK manager. Select the Android 5.0.1 (API 21) checkbox, and click install packages. Accept the license agreement and install. For more information on adding packages, see the following link.

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

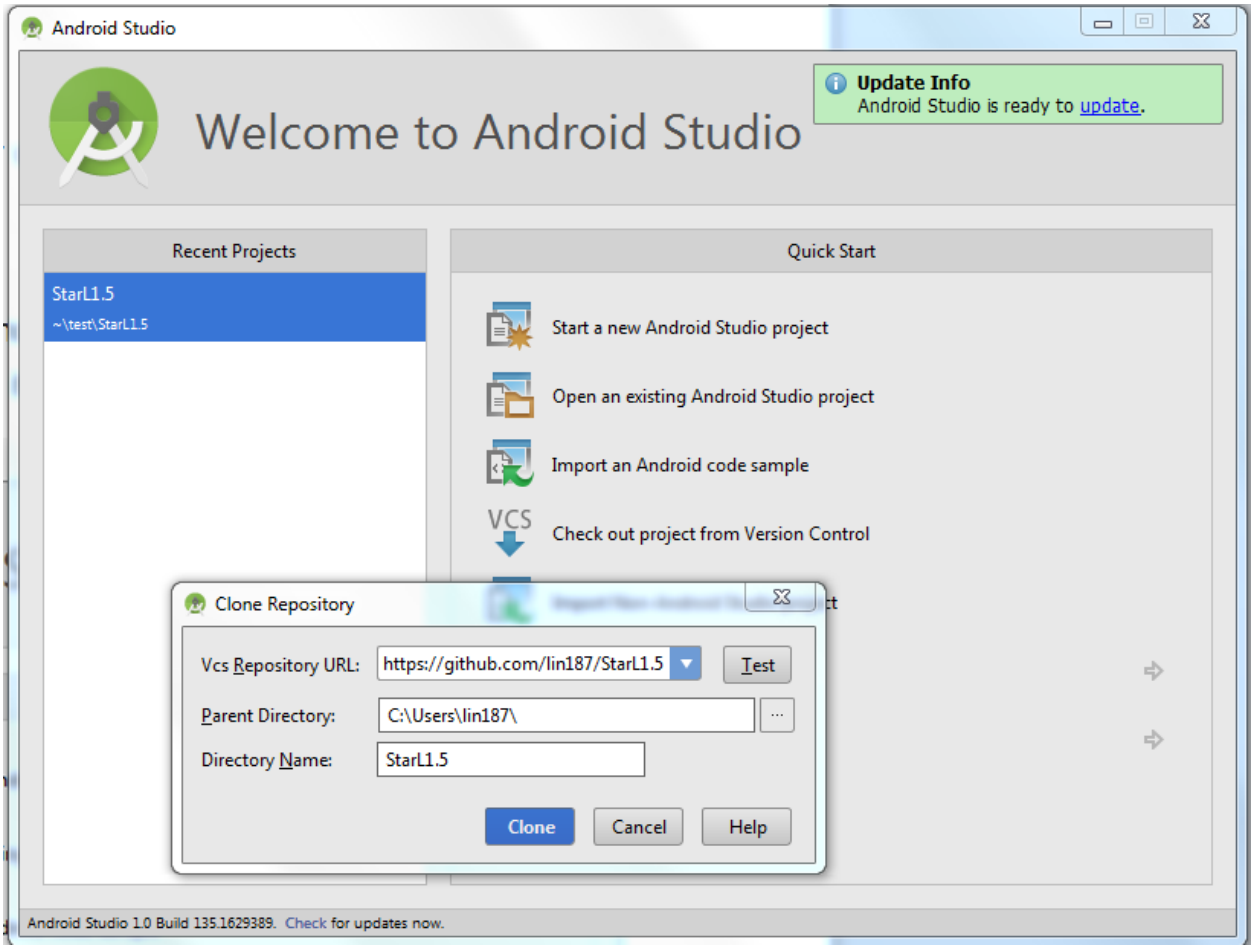
Next, you need to acquire StarL from github. You can either check out the project using version control, or you can download a zip file and import the project into Android Studio.

Option 1 check out project from version control:

If you choose this method, you will need to install git if you do not already have it installed. You can use the following link to install git.

http://git-scm.com/

After installing git, click Check out project from Version Control in Android Studio and select Git from the dropdown menu. Then enter https://github.com/verivital/starl-uta/ as shown in the screenshot below and click clone (note: do not use https://github.com/lin187/StarL1.5/ as we may make updates to StarL over the semester that may not be incorporated in the original repository).



Option 2 download a zip file and import:

Download the zip file using the link below by clicking “Download ZIP”.

https://github.com/verivital/starl-uta

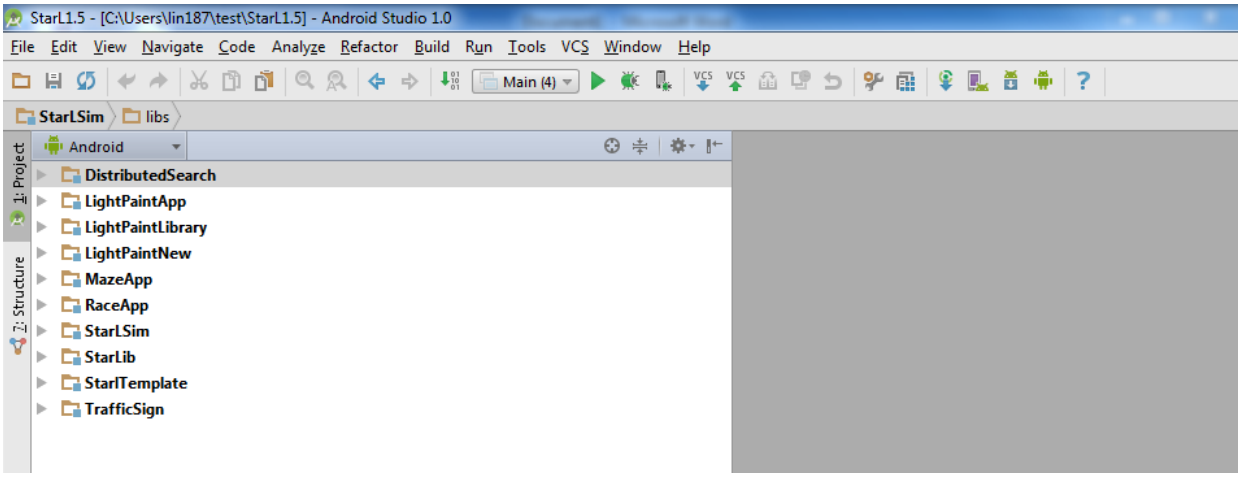
A direct link is here:

https://github.com/verivital/starl-uta/archive/master.zip

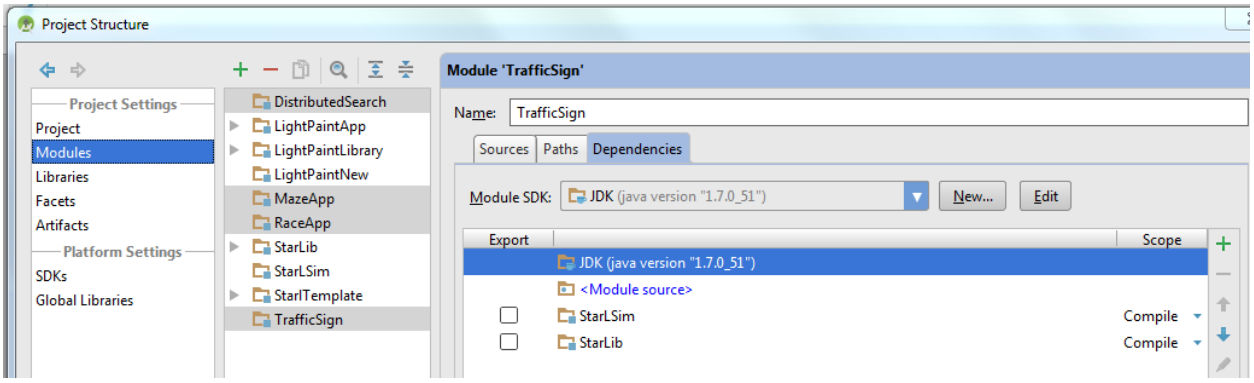
After downloading, unzip the zip file directory. In Android studio, click Open an existing Android Studio project, navigate to and select the unzipped directory, and click OK.

Setting the dependencies:

After opening the project, you should see something similar to the following.



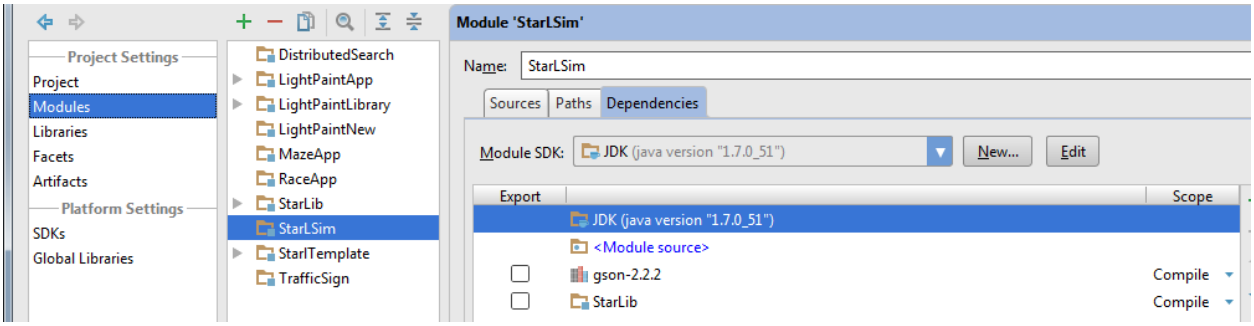
Before running the simulations, click File, Project Structure. Then click on project and set the Project SDK to Android API 21. If the project language level is not set to 7.0, set it to 7.0. Next, select modules from the same window. You will need to change the dependencies as follows. Select DistributedSearch then click the dependencies tab. On the module SDK dropdown menu select 1.7 (java version…). It should look as follows.



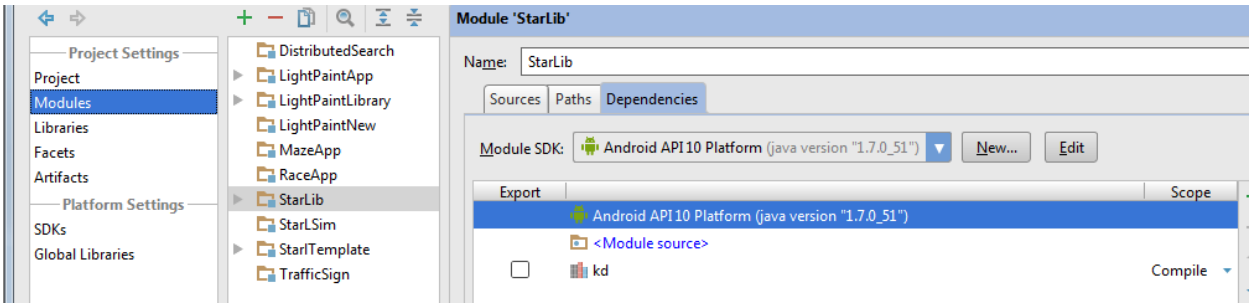
Repeat these steps for MazeApp, RaceApp, and TrafficSign (should look the same as above for these three).

Then change the dependencies for the rest of the folders to look like the images shown for each one. In some of the images, Android API 10 is selected. You should select API 21 instead.

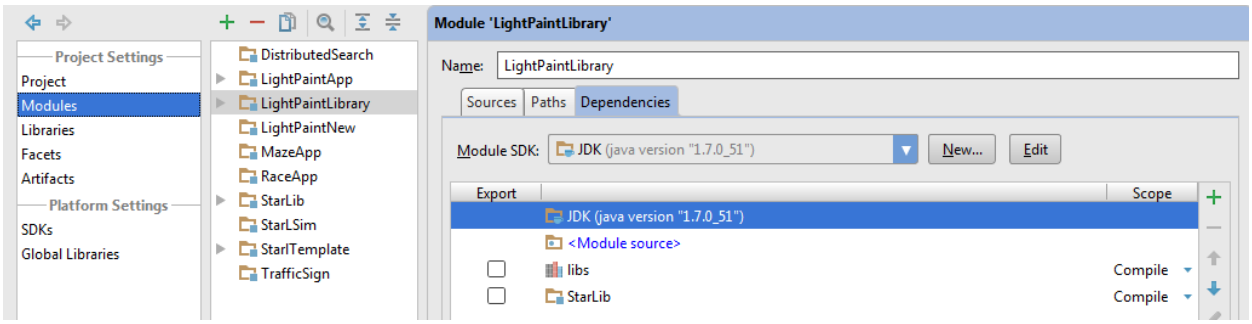
StarLSim :



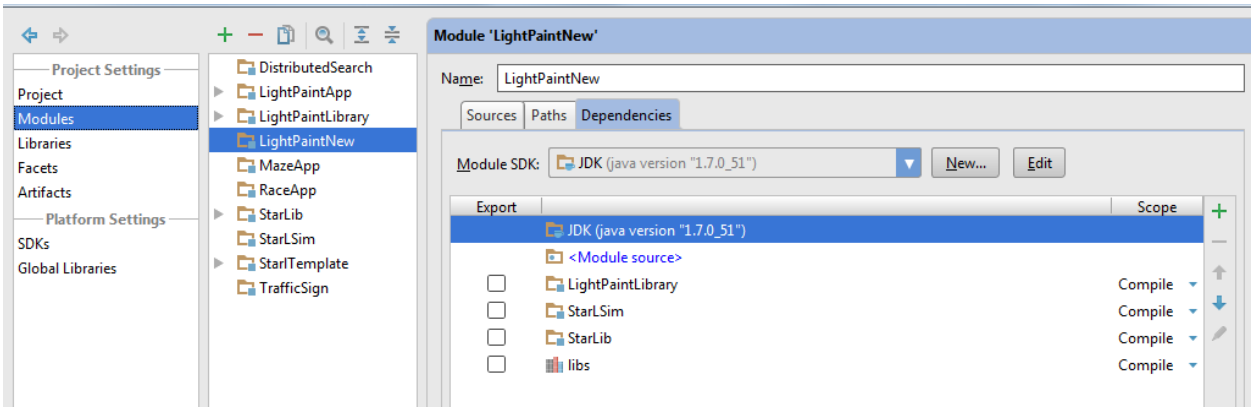
StarLib :



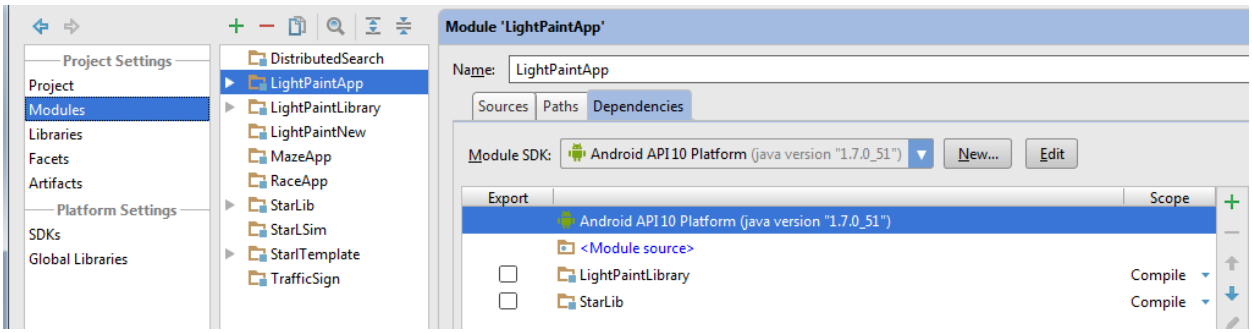
LightPaintLibrary:



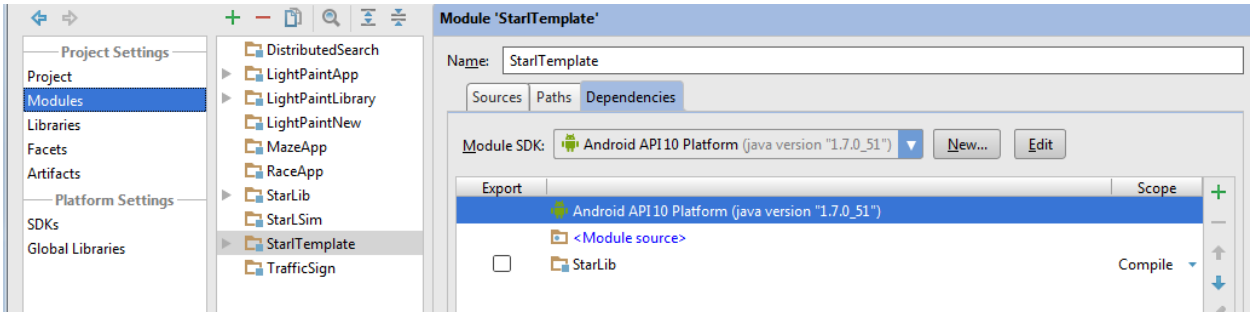
LightPaintNew:



LightPaintApp:



StarlTemplate:



You should now be able to run the simulations as follows.

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:

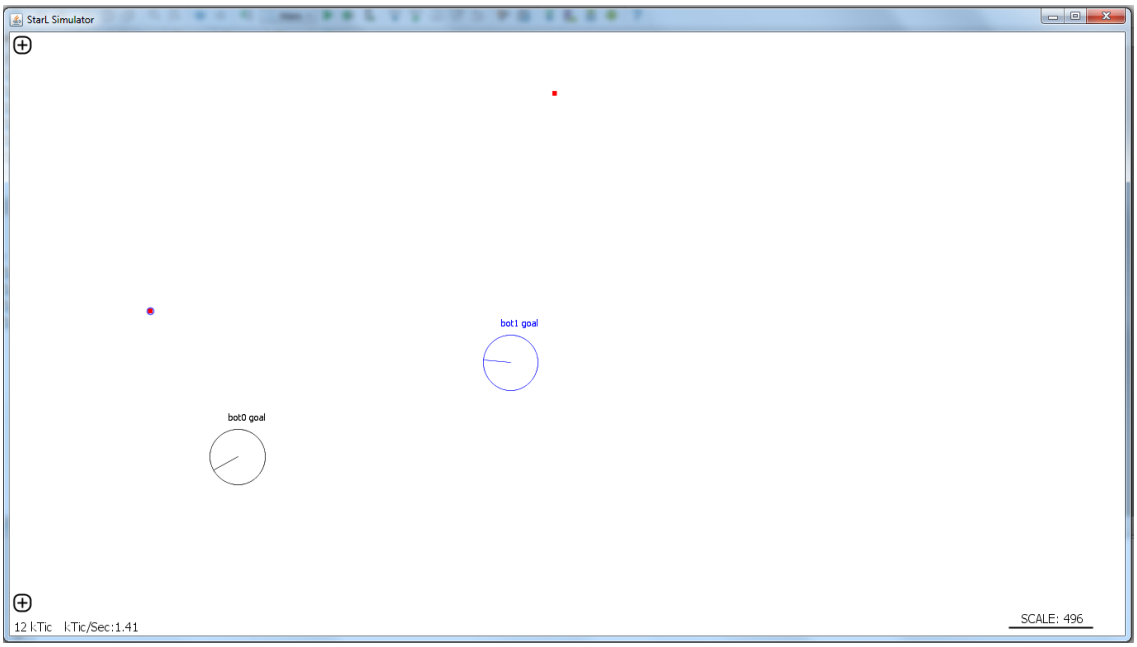
Main file set some values for simulation, for example, number of robots, how fast the time goes. It also creates the simulation application and launches it.

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, right click Main.java under RaceApp source folder. Run Main.main()

You should see something like this:

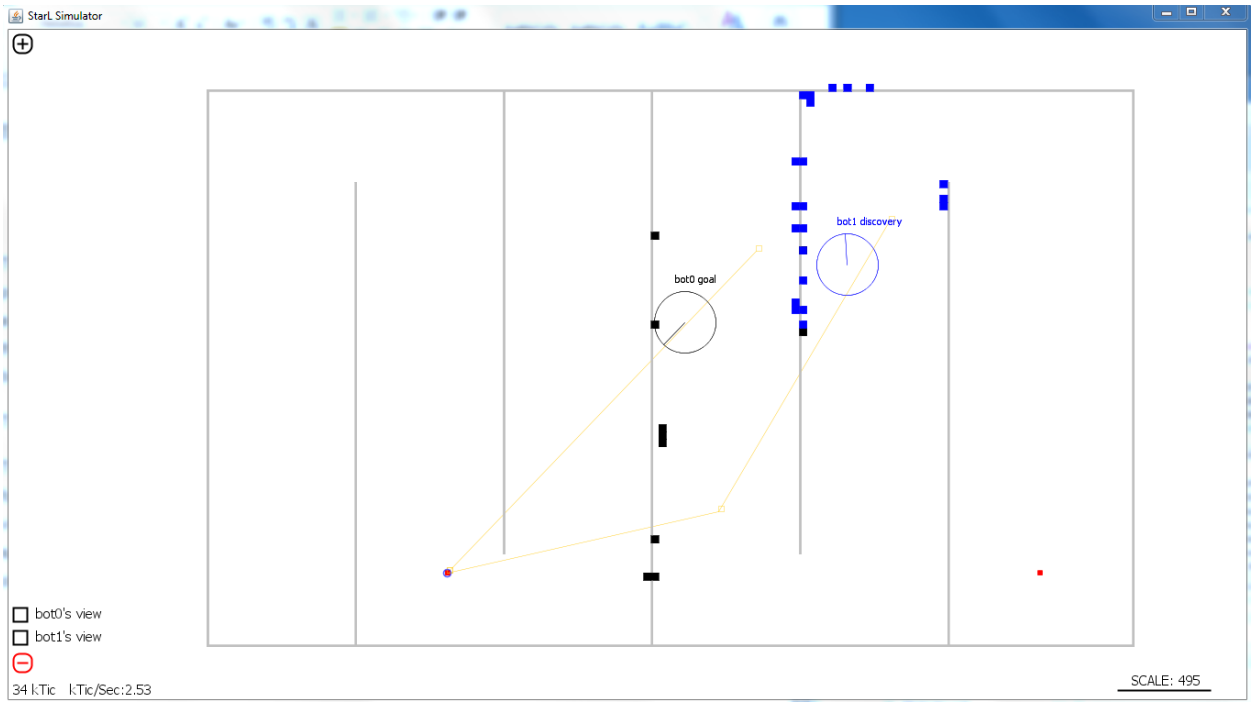


The RaceApp is a simulation where a number of robots (iRobot Create) starting in any point in the map. Their goal is to reach a number of destinations. Once a robot reaches a destination, it announces to other robots. Then they will race to the next point. When the robots collide, they will stop and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

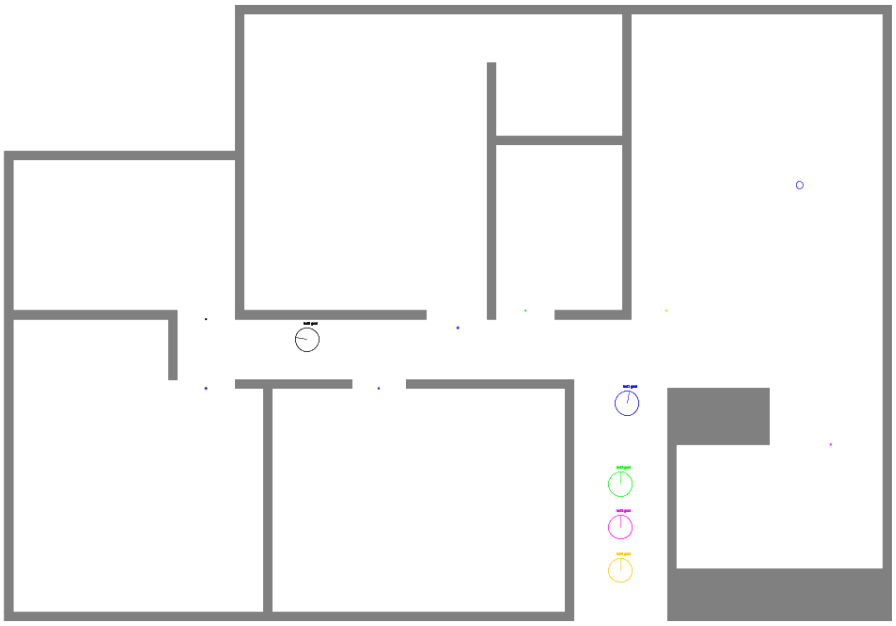
The MazeApp is a simulation for path planning in a maze like environment. Every robot will try to reach a point at the bottom right corner. They will use RRT path planning to avoid colliding with walls. However, initially, they do not know how the maze looks like. They will learn about the position of 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 views are shown in different colors.



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.

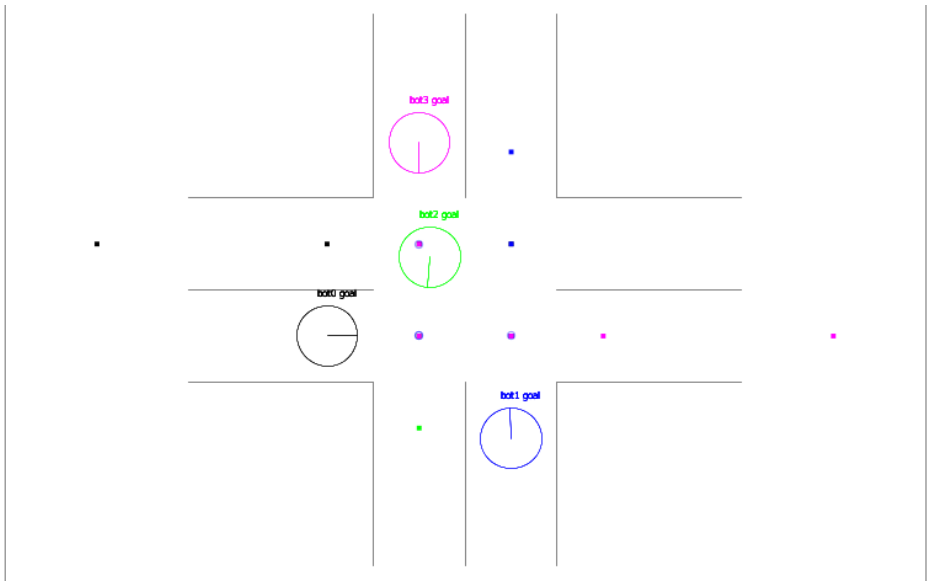
Start.wpt has format of: WAY,x,y,angle,robotName

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

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

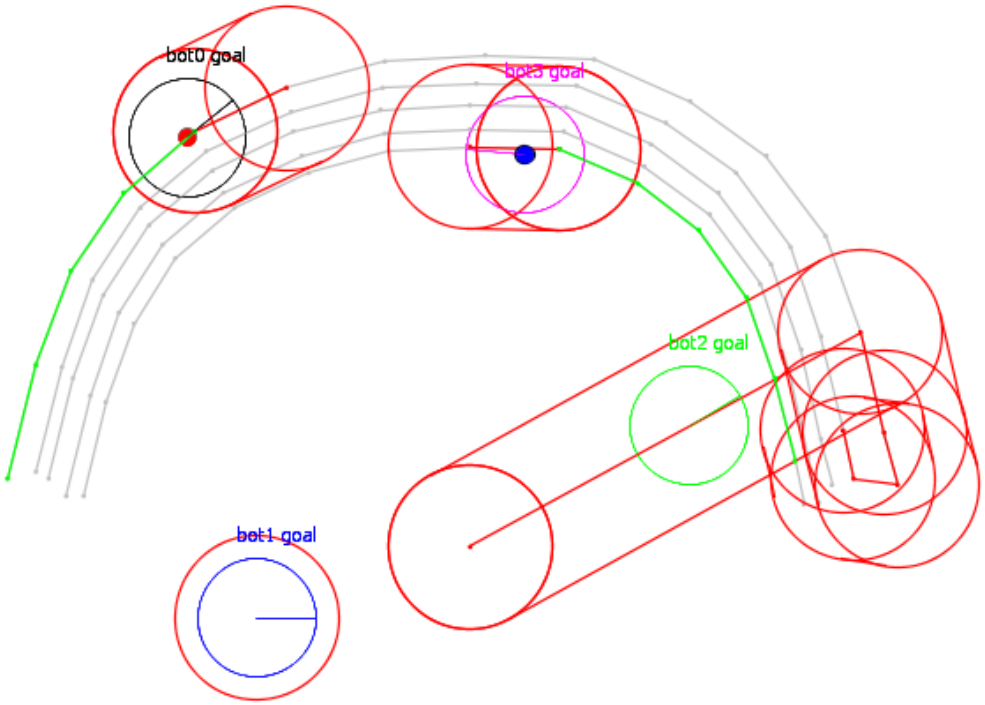
The TrafficSign App simulates a four way stop sign intersection for robots. We implemented a mutual exclusion algorithm so that the robots go through the intersection safely and efficiently. See StarLTrafficSignApp.pdf for additional material on the TrafficSign App.

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:



Setting up StarL with Eclipse ADT:

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

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

Install Eclipse with ADT

https://developer.android.com/tools/help/adt.html

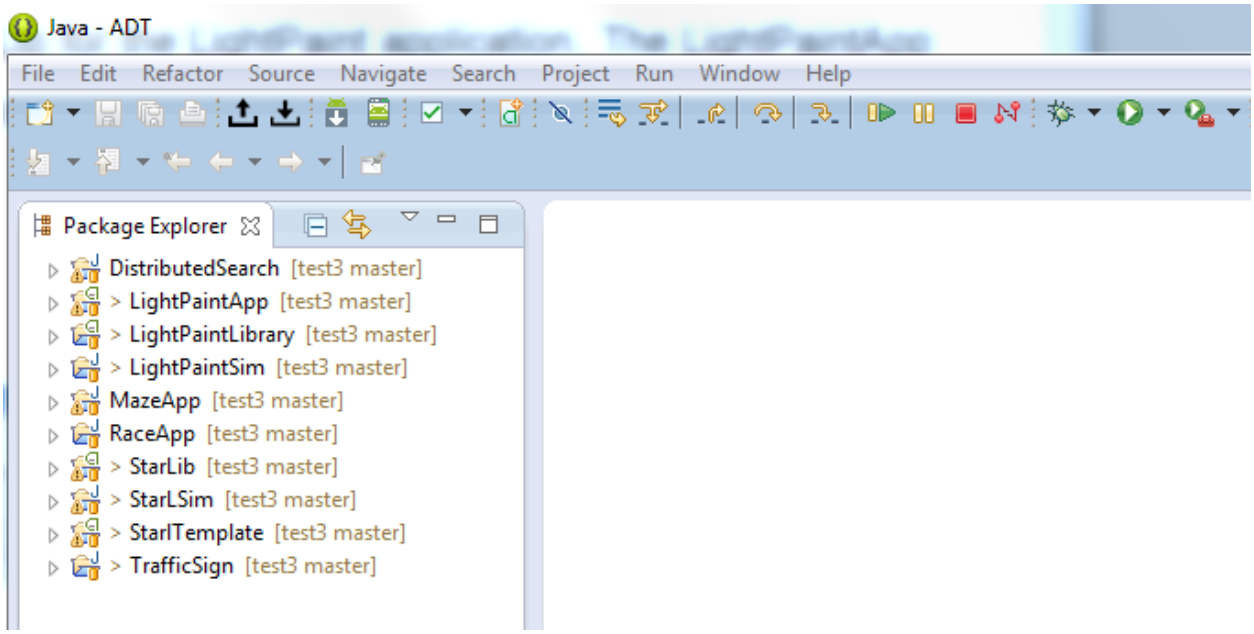
After installing, open Eclipse and click Window, Android SDK Manager. Install the latest SDK(API21) using the SDK manager, if you are having problems, try install SDK 2.3.3(API 10).

Download the zip file from the following link:

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

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.

Your screen should look like:



You will need to set the dependencies. Please refer to the setting the dependencies section above to see what they should be. To set them 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.

To run the simulations, right click Main.java under RaceApp (or other apps) source folder. Run as, Java Application. Please refer to the StarL examples section for more information on running simulations.