Coding with Linkbot and Mindstorms

Technical Document:

Developer's Guide

UC Davis

C-STEM Center

Table of Contents

1 Introduction

This is a guide for developers on how to change and improve the “Coding with Linkbot and Mindstorms” website. This guide will talk about how each part of the website was implemented, what skills will be needed to further develop each of those sections and what files need to be edited.

1.1 Website description

There are four main parts to the website. The first, was the top most toolbar described above. The second part of this website would be the left side portion below the toolbar, which contains the grid. The grid will display the user’s robot of choice (ie. Linkbot or Mindstorms, or both) and its real-time simulated movement based on the blocks used. The distance between the red lines are 12 units, and the distance between each white line is 1 unit. The numbers -12, 0, 12, and 24 are marked along the x- and y-axis at appropriate locations to mark these distances. Directly below the graph are the buttons “Start Over” and “Run.” Clicking the “Start Over” button will delete all blocks currently placed in the workspace (which will be described in more detail next), while clicking “Run” will allow the user to run the code that is generated from the blocks that are placed in the workspace. After “Run” has been clicked, it will change into the “Reset” button, which will allow the user to make changes to the blocks in the workspace and rerun the simulation with the new code. To the right of these buttons, the user can view the robot’s current position (coordinates) and its current orientation angle. Below this, there are buttons that allow the user to select which robot he or she would prefer to use, as well as the lesson prompt. A possible solution is provided if the user clicks the blue “Possible Solution” button, which can be hidden again by clicking on the button again.

The third part of this website is the right side portion below the toolbar. This is the workspace section of the website. On the left side of this workspace section, there are tabs that lead to various blocks. In the example above, they can be found by clicking Robot. In further lessons, there will be tabs available for logic, math, etc. blocks as well. These blocks can be dragged into the white space for use. Extra or unnecessary blocks can

be dragged to the trash can at the bottom right, or back to the left side for deletion. The tabs at the workspace labeled Ch and C++ can be clicked to show the generated Ch and C++ codes respectively for the blocks used in the workspace.

The fourth part of this website is the bottom portion of the website, below the graph and the workspace as shown in Figure 2.

Figure 2: Related links located at the bottom of the webpage

Here a variety of links can be found that will direct you to the UC Davis C-STEM Center's Facebook, Twitter, Flickr, and YouTube pages.

1.2 File Locations

On the local server, the website is saved in the file path: "/home/www_blockly/BlocklyRobotSimulation/"

This includes the entire blockly library and the code for the website. The website code is located in "/BlocklyRobotSimulation/demos/code/". The code folder contains all the code that will be edited for the webpage UI and animation. *index.html* is the first coding lesson and all of the subsequent lessons should link to corresponding html files (i.e. coding lesson 3 is called *c3.html* and math lesson 3 is *m3.html*). The main file to be edited for UI and functionality is *Playground.html* because it has fully functionality. Each lesson (including *index.html*) should follow from playground and have changes based on its specific needs.

2 General UI

2.1 Files

The main files are :

Playground.html

index.html

x[1-n].html (where x is a letter denoting the type of lesson and n is the lesson number. i.e. c2.html)

2.2 Making Changes

Changes are made and tested on *Playground.html* because it has full functionality and can be used as a template for adding new pages. The developer should have a solid background in html/css in order to modify the UI of the website. Style changes are all currently made directly in the html file within each tag, rather than through a css file. The developer may wish to create a stylesheet in order to better organize the way UI is edited.

2.3 Updating Changes to All Lessons

The process of porting changes on playground.html to all other lessons is currently a primitive mechanical process and this can be achieved in multiple ways. Perhaps the simplest way is to copy over the changed portions from *playground.html* and overwrite the same sections in each lesson.

If a stylesheet is implemented in the future, that stylesheet is all that would need to be edited, as all lessons will use it to style its elements. Consequently, id tags will need to be assigned to all sections of the lessons to accommodate for this.

3 Animation

3.1 Files

The main files are :

Playground.html

index.html

x[1-n].html (where x is a letter denoting the type of lesson and n is the lesson number. i.e. c2.html)

3.2 Implementation Notes

The animation consists of the background grid, the robot images, and the movement and manipulation of the robot images. The grid is rendered using svg and the robot sprite is placed on the grid. The robot images are spritesheets of linkbot and mindstorms. The movement is done by adjusting the top and left margins of the robot image.

3.3 The Grid

The grid is drawn using an svg element labeled “id = mysvg”. The pattern with id “smallgrid” controls the smaller rectangles for the grid. Changing these numbers will affect the size of these smaller rectanngles. The pattern with the id “grid” controls the darker lines that represent the axis’s. Changing these numbers will affect the size of the larger, darker rectangles that represent the axis’s).

The grid is labeled using the divs with ids “plus-y”, “minus-y” etc. These values are changed depending on what system of measurement is currently selected; either metric, or US customary units. In order to dynamically change these grid labels when the user changes the units of measurement, we edit the *toolbox()* function to change the unit values based on the units selected.

Due to the way that the grid was is rendered, the animation of the robot is unable to properly disappear when it moves off the grid. Therefore, it is suggested that Canvas is used to render the graph in the future with the robot animation within the Canvas element. This will allow for more flexibility in animations and also make it easier to implement grid tracing for the robots, since there are simpler ways to dynamically draw lines in canvas as opposed to SVG, which is currently what the grid is being drawn with.

3.4 Moving the Robots

The robots are animated by adjusting their top and left margins relative to their starting positions. The coding is done in playground and then ported over to the other lessons, once coding is complete.

The current position (x and y coordinates) are retained globally and so is the angle that the robot is turned , relative to the unit circle. The move function drives the robots forward or backward, and the turn function is used to turn.

Since javascript is asynchronous, set timeouts must be called when any animation blocks are called, so that the blocks execute in a serial manner. NonBlocking functions are an exception and they allow movements to occur simultaneously with other robots.

If and when the grid is replaced with a canvas element, the code for movement must be adapted to fit canvas notation, but the general algorithms should hold true.

3.5 Animation Sprites

The images for the robots are 2d, but they simulate turning in 3d by using spritesheets that capture 360 degrees of rotation. For animation purposes, each image within a spritesheet is 200 pixels wide and tall. The overall spritesheet is 1200 by 2400 pixels. When the turn function is called, the angle of rotation is calculated, and so is the angle the robot should be facing at the end of the animation. The spritesheet only renders 200x200 pixels as a time and slides along the sheet until it reaches the pre-calculated angle (the picture where the robot is facing that angle). This simulates rotation.

4 Workspace

4.1 Files

The main files are :

Playground.html

index.html

x[1-n].html (where x is a letter denoting the type of lesson and n is the lesson number. i.e. c2.html)

code-orig.js

4.2 Workspace UI

There are many ways to implement the workspace UI. These are all designed by Google and there are demos for all the different ways in other subdirectories of the Demos folder. The original developers chose the design with tabs to hide the block as opposed to showing all the blocks in the sidebar. The UI was then adjusted to make the workspace fit better on the screen and contain all the required attributes, such as code generation. The layout of the workspace (height and width aspects) are edited mostly in code-orig.js but can also be edited in playground and the rest of the lessons themselves.

5 Blocks

5.1 Block Creation

The structure of the blocks are located in the folder “blocks,” which contains Javascript files. Several pre-created blocks are located here such as loop blocks or logic blocks. Custom blocks can be created using Block Factory. It is a GUI to form a new block. The link for it is https://blockly-demo.appspot.com/static/demos/blockfactory/index.html. Every block has a distinct name and specific format. The color of the block can be adjusted, it can have inputs of various types, and it can display other fields. The language code should be copied into a Javascript file in the folder “blocks.”

5.2 Code Generation

Now that the blocks’ structure has been created, the code generation needs to be implemented. These files are located in the folder “generators.” Several existing languages have been included from Google Blockly such as Python and Javascript. For “Coding with Linkbot and Mindstorms,” we have added two new languages: Ch and C++. The Javascript file directly located in the folder contains helper functions to generate the code for the blocks. The specific code generation for each individual block is located in the corresponding Javascript file within the language folder in “generators.” Each block contains its own code generation that uses the inputs for the block. Definitions such as include headers or variables can also be added.

5.3 Compiling

The Javascript files need to be compiled into compressed Javascript files located in the main directory. Each language has its own Javascript file for code generation. Compressing Blockly is simple by running the command “python build.py” in the root

directory. One problem that may arise is the error: “Closure not found” which means that you are missing Closure dependency. The files can be obtained from Github and should be placed next to the root directory. There are limitations to how often you can compress Blockly, so compile only when needed.

5.4 Using the Blocks

Now that the blocks are created, they have to be added to the toolbox for the user to drag. In every html file, there is a section labeled <xml id="toolbox" style="display: none"> that contains all of the blocks. Inside this xml file, there are categories depending on how the blocks are organized. For example, the logic category is labeled <category id="catLogic"> with every block in it listed after. The items in the categories do not have to be individual blocks; they can be a combination so the user will have more ease to drag them out together. A loops block has default field inputs included in the toolbar.