Software Design Process

***Move Robot (1 motor)***

* Design Process

1. Motor works
2. Motor works with joysticks
3. Motor works with buttons
4. Motor speed is adjustable

* Components
  + Forward/Backward - 1 Large motor
* Code

***Claw Move (2 servos)***

* Design Process

1. Servo works
2. Servo works with joystick
3. Two servos with two joysticks
4. Two servos with one joystick
5. Servo works with buttons
6. Two servos with two buttons
7. Separate open and close buttons for each servo

* Components
  + Open/Close - 2 servos (both commands)
* Code

***Arm Move (1 motor)***

* Design Process

1. Motor works
2. Motor works with joysticks
3. Motor works with buttons
4. Motor speed is adjustable

* Components
  + Forwards/Backwards - 1 motor
* Code

***Arm Rotate (1 motor)***

* Design Process

1. Motor works
2. Motor works with joysticks
3. Motor works with buttons
4. Motor speed is adjustable

* Components
  + Clockwise/Counter-clockwise - 1 motor
* Code

**Simulation and Debugging**

For simulating and debugging, we hooked up motors and servos directly to the brain and just tested the functionality. We eventually did test the code on the physical robot, but for most of the time, we didn’t have a functional robot yet. We didn’t really have any problems with variables or miscalculations, mostly just problems with code logic.

**Connection to Strategy**

The main goal is to gain points by collecting trash, while at the same time doing more complex tasks like finding a score multiplier, stacking boxes on a structure, or solving a logic puzzle involving rubber ducks labeled 1-4. We first and foremostly decided that moving was our biggest priority. We couldn’t do anything if our robot couldn’t move. We then decided that our second priority was collecting the turtle to increase our score. After that comes collecting trash. Then we started to write code. We figured out movement, extending/retracting/rotating the arm, and finally opening and closing the claw. We needed to make these functions require as little thought as possible, and put them in logical positions on the controller. For example, we grouped the servo control buttons to one part of the controller: the 8D, 8U, 8R, and 8L buttons. We put movement of the drive on channel 4 and control of the arm’s extension/retraction on channel 2 and its rotation on channel 1.

Good software practices:

[insert picture]

* Commenting on code
* Using variables
* Organizing your code

**Error Conditions and Response Actions**

Motors:

Our first real problem with the motors was the fact that they would have choppy movements when we held a button to move them. This would make controlling other parts with motors like the arm and rotation very difficult. The desired result was to have the motors move smoothly and without much resistance. Our diagnosis of the problem was that since we were using if statements, the robot was constantly re-checking for the button, resulting in the choppy movement. Our solution was to use while statements with true/false button conditions to keep it running until the button was released.

The code that helped solve the problem with the motor:

[insert picture of code]

Servos:

We tried to use an equation to toggle the values of position variables using a statement to divide the number of times the button has been pressed by 2 and finding the remainder which will always be either 1 or 0. We turned that into a true/false variable and tried to set the servo positions using that. The result was constant wiggling when the button was held for prolonged periods of time. We couldn’t figure out why this was happening, so we devised a solution. We solved that problem by simply mapping the open and close commands to different buttons instead of trying to toggle it with one.

The code that helped solve the problem with the servos:

[insert picture of code]