OzoBlockly Editor Challenge:
ORA Sorting Shapes Challenge
Grades 9-12
Teacher Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

Welcome to the Teacher's Guide for the "ORA Sorting Shapes Challenge," a comprehensive lab designed to immerse students in the cutting-edge intersection of robotics and Artificial Intelligence. In this activity, students transition from basic coding to advanced automation engineering as they program the ORA robotic arm to sort geometric shapes using a vacuum gripper and AI-driven vision.

This challenge is unique because it integrates a Large Language Model (LLM) to identify objects, mirroring modern industrial applications found in smart factories and recycling centers. Rather than manually defining shape parameters through complex geometry, students leverage the power of computer vision to allow the AI to make autonomous decisions.

As an instructor, you will facilitate a hands-on environment where students bridge the gap between OzoBlockly logic and physical hardware. Key learning objectives include understanding suction-based manipulation, coordinate-based navigation, and the vital implementation of safety protocols including speed management and emergency stops. By completing this lab, students will gain a practical understanding of how global supply chains utilize AI to handle variability, preparing them for the future of the technological workforce.

If you have not already done so, you should read through the *Student Guide* for this challenge for details on what is expected from the students. Also, you are encouraged to view the 11-second time-lapse video that shows ORA in action on this challenge.

Example Solution to this Challenge

The solution to this challenge provides the opportunity for the students to develop an OzoBlockly Editor program that is modular, dividing the program logic into functions each having a specific purpose. In the solution presented here, variable and function names have been created in a way that promotes self-documentation.

Figure 1 shows the main program and the "*Initialization*" function. The initialization function begins by setting ORA's speed to a low value of 20% for safety. ORA is then moved to the HOME location, and the tool is set to the vacuum gripper and turned OFF. The main program begins with a call to the initialization function. Assuming that there are six shapes to sort, a loop is set to repeat six

times. Each time through the loop, three functions are called in succession: *Capture Image Settings, Pick Up the Shape, and Move Shape to Its Proper Bin*. After moving a shape, ORA goes back to the HOME location. When all six shapes have been moved to their bins, ORA moves to the ZERO location.

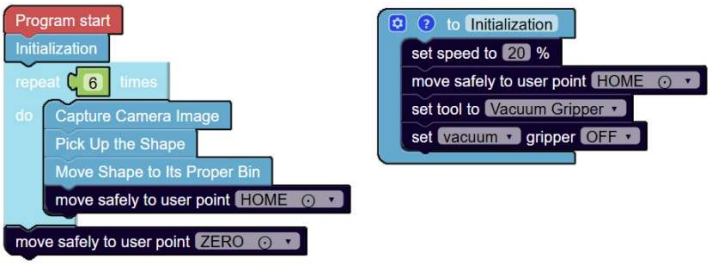

Figure 1

Figure 2 shows the "*Capture Camera Image*" function. The prompt has been designed to return the letter C for a circle, letter R for a rectangle, and T for a triangle. The letter is stored in the variable *shapeType*. The two *print to IO console* blocks are particularly useful when developing and debugging the program. If desired, then can be removed in the final program. The most likely exception is caused by the LLM credits being exhausted and was therefore included in the *in case of exception do* portion of the *try* block.

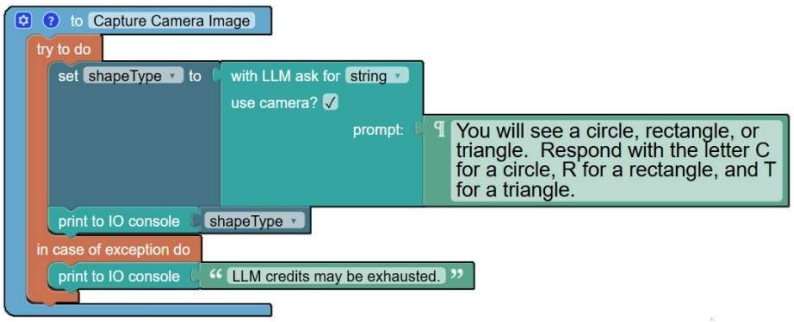

Figure 2

Figure 3 shows the "Pick Up the Shape" function. The pickup location as at (x, y) = (395, 0). A *z* value of 1 mm worked well to place the bottom of the vacuum gripper in contact with cardstock shape. After the vacuum gripper was turned on, it was raised to a height z of 80 mm.

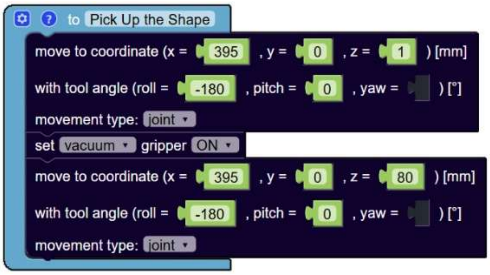

Figure 3

Figure 4 shows the "*Move Shape to Its Proper Bin*" function. Nested *if-do-else* blocks handle the logic of assigning the proper bin location to each of the three shapes. The bins are located as follows: circles at (x, y) = (200, -200), rectangles at (200, 0), and squares at (200, 200). The cardstock shapes are dropped from a height of 10 mm for all three bins.

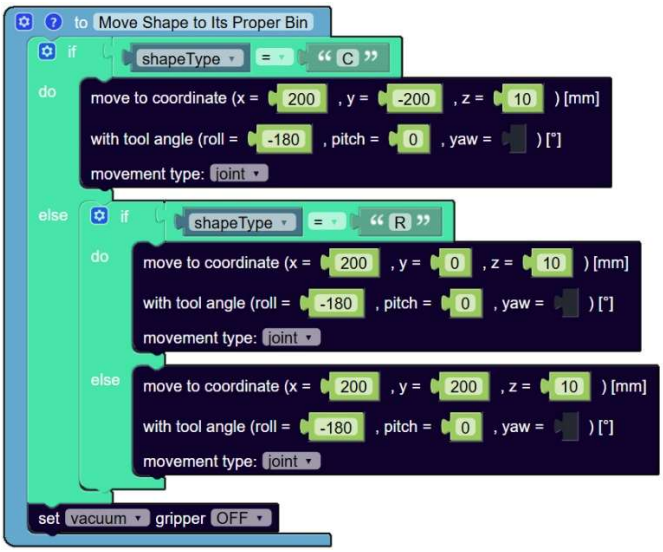

Figure 4

Share Code for this Lesson – 2jn59jw

Zip File – Square Riser for the Cardstock Shapes

ISTE Standards (International Society for Technology and Education)

Empowered Learner 1.1.d: Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.

Knowledge Constructor 1.3.d: Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Computational Thinker 1.5.c: Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

CSTA (Computer Science Teachers Association) Standards

2-CS-02 Design projects that combine hardware and software components to collect and exchange data.

2-AP-11 Create clearly named variables that represent different data types and perform operations on their values.

2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
2-AP-19	Document programs in order to make them easier to follow, test, and debug.
3A-DA-11	Create interactive data visualizations using software tools to help others better understand real-world phenomena.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3B-CS-02	Illustrate ways computing systems implement logic, input, and output through hardware components.
3B-AP-08	Describe how artificial intelligence drives many software and physical systems.
3B-AP-14	Construct solutions to problems using student-created components, such as procedures, modules and/or objects.