OzoBlockly **Intelligent Currency Challenge**

Grades 9-12

Teacher Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

The **ORA Intelligent Currency Challenge** provides your students with a sophisticated intersection of robotics, artificial intelligence, and financial logic. Designed as an advanced extension of OzoBlockly, this challenge transitions students from basic movement commands to the integration of 6-axis kinematics and Large Language Model (LLM) vision systems. This guide is intended to help you facilitate an environment where students can master the complexities of real-time data processing and conditional automation.

As an instructor, your role is to guide students through the synthesis of three critical domains: computer vision, precise robotic manipulation, and cumulative logic. The project centers on a "*Smart Deposit*" system, requiring the ORA cobot to not only recognize varying coin denominations—pennies, nickels, dimes, and quarters—but also to maintain a running digital total. The challenge concludes only when the system reaches or exceeds a specific financial threshold of one dollar. This mimics industrial automation found in retail and logistics, where robots must make autonomous decisions based on visual input.

Success in this lab relies heavily on environmental calibration. You will need to assist students in optimizing the lighting to ensure the LLM can distinguish coin features and in setting up the physical workspace to accommodate the manual transfer of coins from the camera station to the pickup location. Safety is paramount. This guide emphasizes the use of ORA speed limiters within OzoBlockly and the constant availability of the Emergency Stop Button. By completing this challenge, students will gain practical experience in debugging complex systems and understanding how AI can be leveraged to solve quantitative, real-world problems.

As mentioned in the student guide, you should view the accompanying 20-second time-lapse video that shows coins being dropped until the one-dollar goal has been reached. There is also a 2-minute video at standard speed that shows the first three coins being identified and dropped into the collection cup. This video makes it easier to see the manual and ORA movements of the coin.

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 the variable *totalCents*, which keep track of the total money dropped into the smart collection cup, to zero. The remaining blocks in this function are self-documenting. Note that the speed is set to a low value of 20% for safety.

The main program begins with a call to the initialization function. A *repeat while* loop is run as long as *totalCents* is less than 100, meaning that the loop will stop as soon as totalCents is 100 or has just exceeded 100. Each time through the loop, three functions are called in succession: *Capture Camera Image, Pick Up the Coin, and Move Coin to Collection Cup*. As soon as *totalCents* equals or exceeds 100, the vacuum gripper is turned off and ORA's arm is moved to the ZERO position.

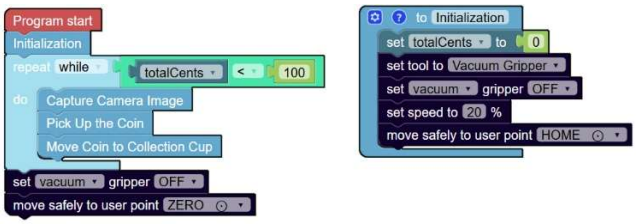

Figure 1

Figure 2 shows the "*Capture Camera Image*" function. The prompt has been designed to return the letter P for penny, N for nickel, D for dime, and Q for quarter. The letter is stored in the variable *coinDenomination*. The *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. A nested *if-else-if* block assigns the proper numerical value of the coin to the variable *thisCoinCents*. Then the variable *totalCents* is increased by the value of *thisCoinCents*.

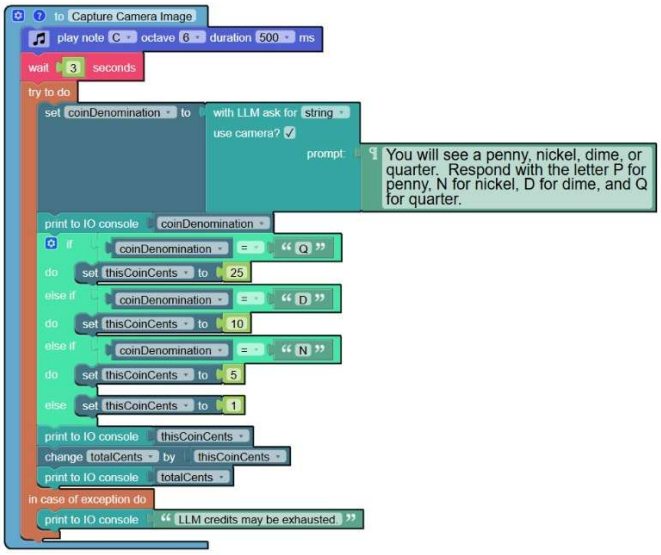

Figure 2

Figure 3 shows the "*Pick Up the Coin*" function. The pickup location is at (x, y) = (395, 0), and at an initial height of 100 mm. A *z* value of -3 mm worked well to place the bottom of the vacuum gripper in contact with coin. After the vacuum gripper was turned on, it was raised back to a height *z* of 100 mm.

Figure 3

Figure 4 shows the "*Move Coin to Collection Cup*" function. The cup is located at (x, y) = (200, 200). The coins are dropped from a height of 100 mm into the collection cup. After the coin is dropped, the vacuum gripper is turned OFF and then there is a 1 second wait. The purpose of the *wait* block is to allow the dropped coin to settle down before Evo says the value of the coin followed by the value of all coins so far in the collection cup.

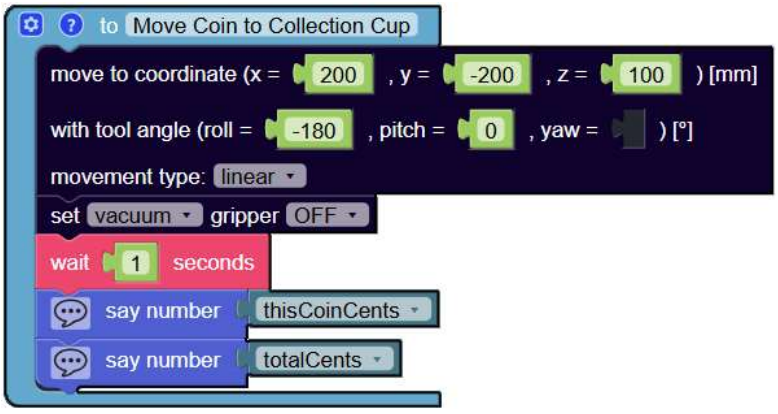

Figure 4

Share Code for this Lesson – xuutahc

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.