OzoBlockly Editor Challenge:

Precision Placement – ORA Builds a Semicircle

Grades 9-12

Teacher Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

Welcome to this OzoBlockly Editor Challenge. This engaging project not only enhances students' programming skills but also reinforces critical mathematical concepts relevant to engineering and robotics. In this challenge, students will navigate the intricacies of the Cartesian coordinate system and apply trigonometric functions, specifically sine and cosine, to execute precise geometric placements of square blocks into a semicircle formation.

As an educator, your role is pivotal in guiding learners through this hands-on experience that blends theoretical knowledge with practical application. The challenge encourages students to think critically and problem-solve like professional engineers, honing skills vital for futures in technology and automation.

By facilitating this project, you will help students understand how abstract math translates into real-world scenarios, fostering creativity and innovation. You'll also find essential safety guidelines and setup instructions to ensure a smooth and effective learning experience. Get ready to inspire your students as they embark on this exciting engineering journey, bridging the gap between theory and practice!

If you have not already done so, you should now read through the *Student Guide* for this challenge for details on what is expected from the students and view the 25-second accompanying time-lapse video.

Special Extension to This Challenge

In addition to having ORA Build the semicircle, you could have your students add "spokes" and make it look like a half of a wheel instead of just a semicircle. This can be accomplished by adding a function to the program that creates the spokes. If your students have a good handle on the math required by this challenge, the function is quite straight forward and contains only a few blocks.

The final result is shown in Figure 1. You can clearly see that if it were possible to extend the spokes, they would all meet at the center of ORA. Of course, they should not be extended, as doing so could result in ORA having a self-collision with possible damage to the cobot. The final section of the student guide outlines some safety measures that help keep this from happening.

Figure 1

Share Code for the Solution – he67mfk
Link to 3D STL File – Qwirkle Square Holder

Example Solution to this Challenge

The solution to this challenge provides the opportunity for your students to develop an OzoBlockly Editor program that is modular, dividing the program logic into functions each having a specific purpose. Images of the modules follow with a discussion explaining the purpose of the Ozobot Editor blocks that make up each module. A separate included PDF file also contains images of these modules, in the event that you want to share any of them with your students.

Figure 2 shows the main program and the "*Initialization"* function. The top-down approach simply divides the logic into three main functions: "*Initialization*", "*Create Semicircle"*, and "*Create Spokes"*. The "*Initialization*" function, shown to the right of Figure 2, contains ORA blocks that are self-explanatory. A speed of 20% or less is encouraged, particularly when developing and debugging the program.

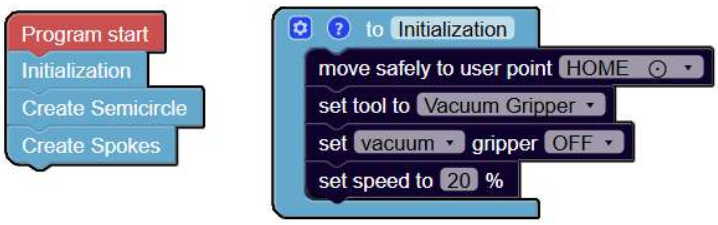

Figure 2

To better understand the "*Create Semicircle*" and "*Create Spokes"* functions, to be discussed next, you may find that a discussion of Figure 3, would be helpful to your students. This figure

clarifies the mathematics, including ORA's Cartesian coordinate system, the trigonometry involved in the solution to the challenge, and conversion from polar to Cartesian coordinates.

The front of ORA contains the Cartesian coordinate quadrants commonly identified by Roman numerals **I** and **IV**, with the x and y axes as shown in white in the figure. The origin of the coordinate system is at the center of ORA's base. The angle θ (theta), shown in light blue, is the angle between the *x* axis and any radial line, shown in dark blue, extended from the origin. The angle θ is positive in quadrant **I** and negative in quadrant **IV**. The Cartesian coordinates *(x, y)* can be obtained from the polar coordinates *(r, θ)* by the formulas $x = r \cos \theta$ and $y = r \sin \theta$.

It should be noted that angles in the Ozobot Editor are, by default, measured in degrees. This is advantageous, as many students are likely to be more familiar with angles in degrees than angles in radians. Angles in degrees are also more intuitive as students have studied angles in degrees back when they took geometry courses.

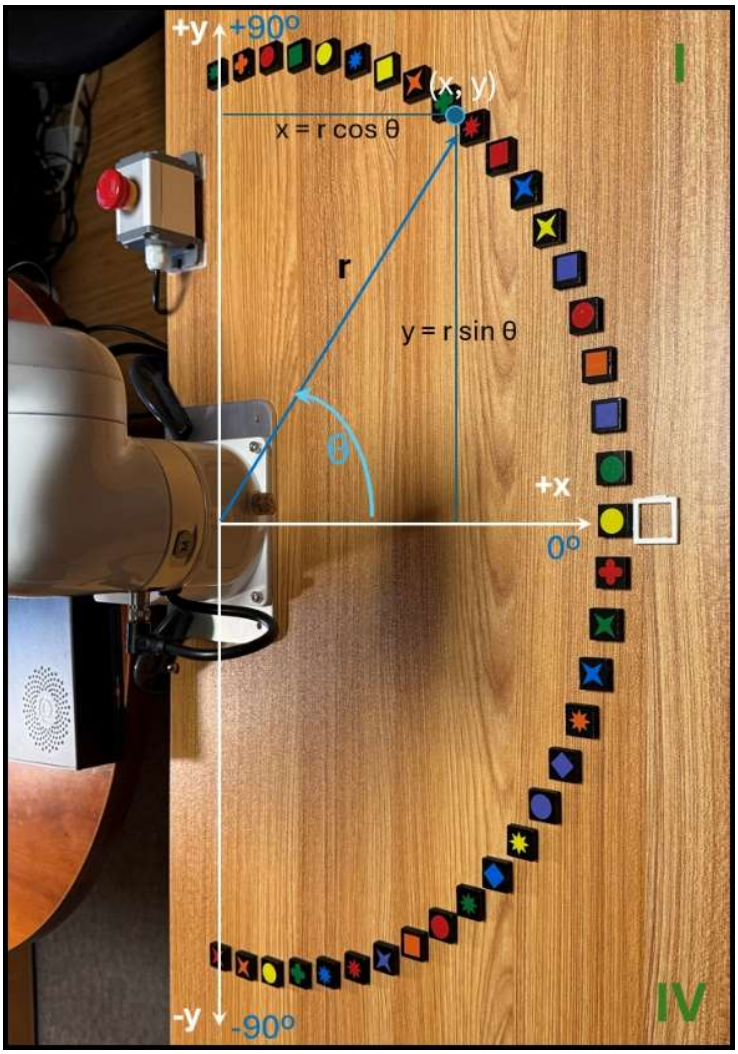

Figure 3

Figure 4 shows the *"Create Semicircle"* function. The radius of the circle was set to 350 mm. The start and end angles are set to -90º and 90º, respectively. Squares are placed 5º apart, which is about the smallest possible angle before the squares begin to overlap, which must be avoided. A simple *count with* loop varies the angle theta from the start to the end angles with increments set as indicated. The loop contains calls to two functions in the order "*Pick Up Square"* followed by "*Place Square"*. The squares are picked up at the location of the 3D printed square holder shown at the right center of Figure 3. They are then placed at the appropriate location on the semicircle.

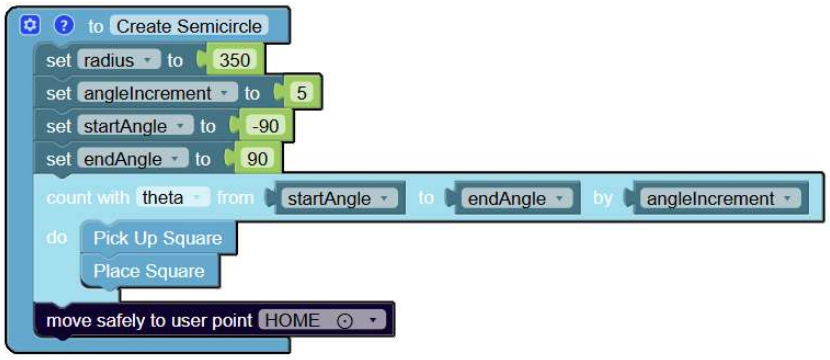

Figure 4

Before studying the "*Pick Up Square"* and *"Place Square"* functions, let's take a look at the "*Create Spokes"* function shown in Figure 5. As you can see in the figure, this function also makes use of the "*Pick Up Square"* and *"Place Square"* functions. A double-nested loop creates one spoke at a time, starting at -90º through +90º, by steps of 30º. Each spoke has a square at radii that vary from 325 mm to 175 mm by steps of -30 mm.

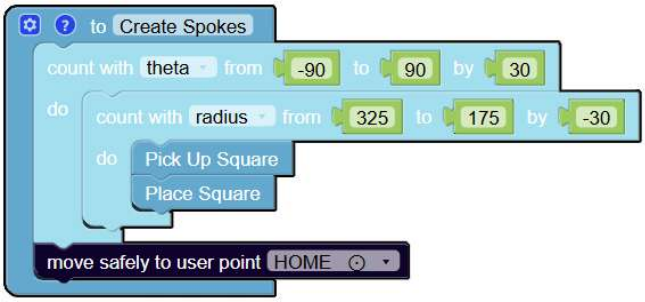

Figure 5

Figure 6 shows the "*Pick Up Square*" function. This function directs ORA to pick up the square that is currently in the 3D printed square holder at (*x, y*) = (375, 0). The first *move to coordinate* block moves ORA's vacuum gripper to a height of 30 mm above the pick-up location. The second block lowers the gripper so that it contacts the square at a height of 3 mm. The gripper is then turned on, and the square is lifted back to a height of 30 mm above the pick-up location.

Figure 6

Figure 7 shows the "*Place Square*" function. The first *move to coordinate* block moves the square at a height of 30 mm to the point (*x, y*) represented by the polar to Cartesian formulas discussed earlier. The second *move to coordinate* block lowers the square to the drop height, and then the vacuum is turned off to release the square. The third *move to coordinate* block raises the gripper back to a height of 30 mm above the tabletop.

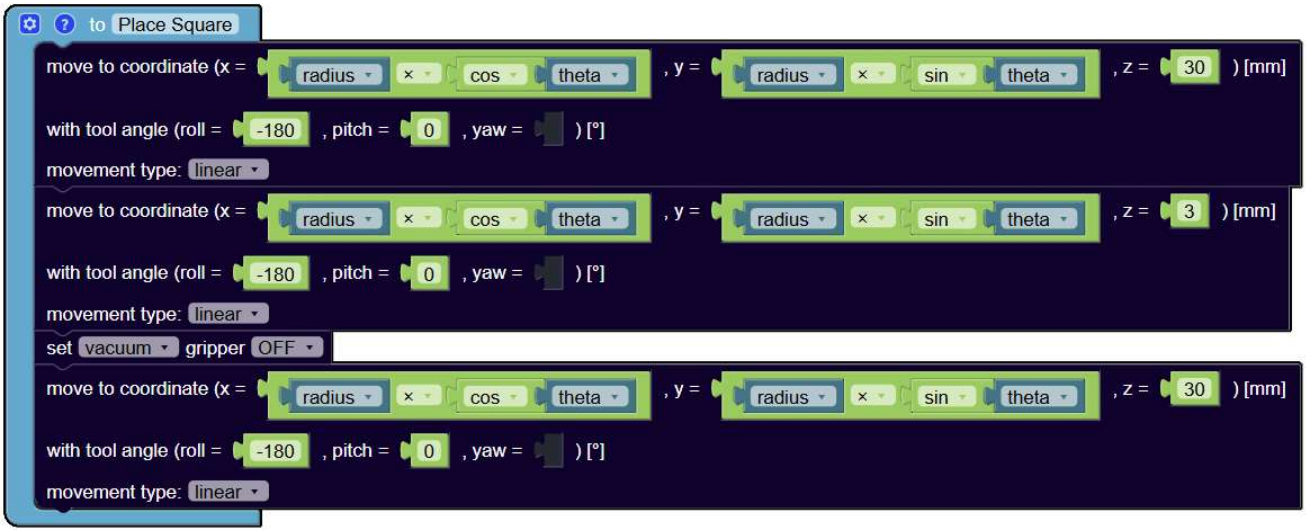

Figure 7

By now, the top-down nature of this solution to the challenge should be clear. Figure 8 provides a hierarchy chart that shows the main program and all of its functions. The arrows may be read as "**calls the** ... **function**". For example, the *Main program* **calls the** *Initialization* **function,** as well as

the *Create Semicircle* function and the *Create Spokes* function. Both of those functions, in turn, call the *Pick Up Square* and *Place Square* function.

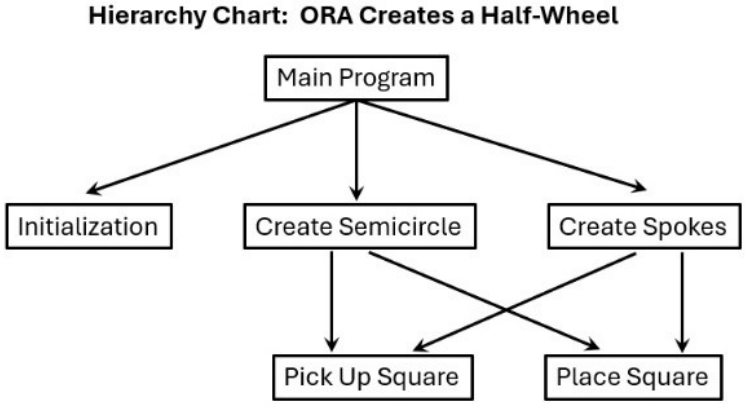

Figure 8

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.