Exercise 1: Team Robot

A common pattern for computer programs is to get input and process it to generate some desired output. Many systems can also be modelled in this manner, including robots. A simple robot has sensors that gather input. The input is passed to a controller that processes it and determines what output to do. The output directs manipulators to change the robot’s environment to achieve some desired goal.

For this exercise you will work with a team to simulate the parts of a robot. Your robot team will be asked to arrange items (e.g., straws and plates) into specific goal configurations (e.g., the green straw is on top of the blue plate). Your robot team includes a brain, eyes, and hands:

• **The Brain** (processor/controller). One member of the team plays the *brain*. Only the brain can see a picture of the goal configuration. The brain can ask simple questions of the *eyes* (questions to which only short answers are needed). The brain can also tell the *hands* what to do and again should try to keep the commands simple. The brain cannot see the work area where the hands are arranging the items or ask the hands any questions.

• **The Eyes** (input/sensors). One member of the team plays the *eyes*. Only the eyes can see the work area. The eyes can only respond to the brain’s questions and should try to keep the answers short and simple. The eyes cannot see the picture of the goal configuration. The eyes cannot ask any questions or speak to the hands.

• **The Hands** (output/manipulators). One or two members of the team play the *hands*. Only the hands can move items in the work area. The hands respond to the brain’s commands. The hands cannot see or speak (but can listen to the brain).

**Part (a)**. Divide into groups of 3 or 4. Decide who will be playing which part. Make sure the brains can’t see the work area, and that the hands have put on their blindfolds. Your Team Leader will now show the brains a picture of the goal configuration to be built from the items in the work area, and the eyes will empty the contents of the bag onto the work area. Your team should try to make the goal configuration as quickly and accurately as possible. After a certain amount of time your Team Leader will say stop and reveal the goal configuration to everyone.
Part (b). Have the members on your team try playing different parts of the robot to build another (different) goal configuration.

For Discussion:
1. Which of the parts was hardest to play? Why? Which was easiest? Why?
2. What type(s) of item configurations were hardest or easiest to arrange? Why?
3. What kind(s) of questions/commands were most useful? Why?
4. What kinds of behaviors of team members were helpful? Not so helpful (be nice)?

Part (c) (optional). Have a competition among teams to see who can reach the goal most quickly and accurately. Develop a scoring system to judge the results.

Exercise 2: Picasso (Java Programming)

A Java program involves creating and manipulating objects, each of which provides some operations. An operation can either perform a task (like printing something on the computer screen), or it can do a computation and tell you the answer. Some operations require that you provide values to be used in their task/computation.

For this exercise, we’ll assume that we have an Artist object named picasso that draws in a two dimensional grid of cells. It provides the operations listed on page 5. Each operation performs a task:

Part (a). What is drawn when the following code executes? (Use the grid to do the drawing; assume that the current position starts in the top left corner of the grid.)

```java
picasso.drawLineDown( 7 );
picasso.moveUp( 4 );
picasso.drawLineRight( 2 );
picasso.moveUp( 3 );
picasso.drawLineDown( 7 );
picasso.moveRight( 2 );
picasso.drawLineRight( 2 );
picasso.moveLeft( 1 );
picasso.drawLineUp( 7 );
picasso.moveLeft( 1 );
picasso.drawLineRight( 2 );
```
**Part (b).** Divide into groups of two. Each group choose one of the pictures shown below and write Java code that would make *picasso* draw the picture. Assume that the current position starts in the top left corner of the grid. (The dotted lines are not part of the pictures; they just show you the cells in the grids.)

![Grids with dotted lines showing cells.](image)

**Part (c).** Now divide into three groups, each using one laptop. In the grid below, draw your own picture and then write Java code for just a small part of your drawing. Enter the code on the laptop and run the program to see the result. Once you get a part of your drawing working, do another part, then another and another until you’ve completed the program for your drawing. This is called *incremental development.*

![Grid with dotted lines showing cells.](image)
For discussion:

1. What other Artist methods would have made it easier to draw the pictures in Part (b)?

2. For Part (a), you traced out by hand what a piece of code would do. Does this seem like it might be useful when you write Java programs? Why or why not?

3. Is there only one right answer for Part (b)? What makes one answer better than another?

4. Most programming languages, including Java, let you write loops. For example, in Java you can essentially say “repeat the following commands n times”. Can any of the code for the pictures in part (b) be simplified by using loops?

5. Why do you think incremental development, like you used in Part (c), might be useful when you write Java programs?
**Artist Operations**

drawLineDown( int length )
Draw a vertical line starting from the current position and going straight down for the given number of cells. The current position is changed to be at the bottom end of the line.

drawLineUp( int length )
Draw a vertical line starting from the current position and going straight up for the given number of cells. The current position is changed to be at the top end of the line.

drawLineRight( int length )
Draw a horizontal line starting from the current position and going straight to the right for the given number of cells. The current position is changed to be at the right end of the line.

drawLineLeft( int length )
Draw a horizontal line starting from the current position and going straight to the left for the given number of cells. The current position is changed to be at the left end of the line.

moveRight( int d )
Move the current position d cells to the right.

moveLeft( int d )
Move the current position d cells to the left.

moveUp( int d )
Move the current position d cells up.

moveDown( int d )
Move the current position d cells down.
Exercise 3: Logical Thinking (KenKen)

One of the benefits many students find they get from taking Computer Science courses is that it helps develop their logical-thinking skills. We’ll work on that today by trying some Ken Ken puzzles (which are similar to Sudoku).

Here’s an example:

The object is to fill in the squares with the numbers from 1 to 4 so that
• the same number doesn’t appear twice in any row or any column, and
• the numbers in each heavily outlined set of squares (called cages) combine (in some order) to produce the target number in the top corner of the cage, using the mathematical operation indicated.

Cages with just one square should be filled in with the target number in the top corner. It’s OK to have the same number more than once within a cage as long as it is not in the same row or column.

Work with a partner to solve the puzzle given above. To solve the puzzle you could just guess, but that won’t work very well. Instead, see if you can find 3 squares that are forced to have a particular number in them. Wait until both you and your partner have found 3 squares like that, then explain to each other which squares they are, and why you think they need to have the numbers you put in there. If you think your partner made a mistake, explain why (nicely!). Then work together to solve the whole puzzle.

Once you’ve finished, try some of the puzzles on the next pages. (For a 6-by-6 grid, use the numbers from 1 to 6.)
<table>
<thead>
<tr>
<th>11+</th>
<th>1−</th>
<th>6</th>
<th>4x</th>
</tr>
</thead>
<tbody>
<tr>
<td>5−</td>
<td>15x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6+</td>
<td>7+</td>
<td>15x</td>
</tr>
<tr>
<td>8+</td>
<td>10x</td>
<td>5−</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2/</td>
<td></td>
<td>18+</td>
</tr>
<tr>
<td>4</td>
<td>6x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3/</th>
<th>7+</th>
<th>10+</th>
<th>6</th>
<th>2/</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3−</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25x</td>
<td></td>
<td>1−</td>
<td>2/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14+</td>
<td>3+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1−</td>
<td>12x</td>
<td>14+</td>
<td>10x</td>
<td></td>
</tr>
</tbody>
</table>