|  |  |
| --- | --- |
| **AP Computer Science GridWorld Case Study (GWCS)** | |
| **GridWorldLab11** | **Sorting Cells with SortCritter** |

|  |
| --- |
| **Lab Objectives** |
| Demonstrate knowledge of a fundamental sorting technique.  Integrate a sorting algorithm in the act method of a Critter object. | |

|  |
| --- |
| **Lab Prerequisites** |
| **Completed ExpoJava, Chapter 18, and completed GridWorldLab10**.  Have a clear understanding of the ArrayList class and its methods.  Understanding fundamental sorting algorithms, like the bubble sort. | |

|  |  |
| --- | --- |
| **Lab Sequence of Steps** | |
| **#** | **Actions** | | **Comments** |
| **01a** | **Setting up GridWorldLab11**  Create Project **GridWorldLab11**  Compile and execute the project.  In **Figure 01** you will see the start of the execution.  **Figure 01** | | This happens automatically when the number of rows and columns are increased.  If you look in the **Lab11Tester.java** file, you will see where the numbers or rows and columns are specified.  If the number of rows and columns are both less than 19, the cells are normal size.  If the number of rows and/or columns are between 19 and 35, the cells are half size.  If the number of rows and/or columns are greater than 35, the cells are quarter size. |

|  |  |
| --- | --- |
| **01b** | **Setting up GridWorldLab11**  Maximize the window to see all of the rows and columns as in **Figure 02.**  **Figure 02**    For this assignment, we have 22 rows and 35 columns for a total of 770 half-size cells. |

|  |  |  |
| --- | --- | --- |
| **02a** | **Objective #1**  You will work in the **main** method of the **Lab11Tester.java** file.  Fill in the entire grid with **Cell** objects.  Each **Cell** object will have a random shade of red as in **Figure 04** below.  **Figure 04** | **Cell** objects inherit from the **Actor**.  A **Cell** object is just a square with a random shade of red that does nothing.  **Cell** objects automatically have a random shade of red once they are created.  You may wish to look at the electronic version of this lab assignment on your computer. The black-and-white paper assignment probably is not that clear. |
| **02b** | **Add the SortCritter**  Place a **SortCritter** object in the top-left hand corner (row 0, column 0) as in **Figure 05** below.  **Figure 05** | **SortCritter** objects inherit from the **Critter** class.  Right now, if you click *Run* nothing will happen. This is because there is nothing in the **SortCritter**’s **act** method. |

|  |  |  |
| --- | --- | --- |
| **03** | **Objective #2**  The second objective of the program is to arrange the cells from darkest to brightest. This is done left-to-right and top-to-bottom.  Each *Step* of the execution places one **Cell** in the correct location.  **Figure 06** shows the completed screen that is completely sorted.  **Figure 06** | You are essentially writing a *Bubble Sort*. Every time *Step* is clicked, the inner loop of the *Bubble Sort* is executed, which puts the next “largest element” – or in this case – “brightest red” in the correct position.  After 768 *Steps* the **Cell**s should be sorted from darkest to brightest.  NOTE:  **Actor** objects have a **getColor()** method.  **Color** objects have methods **getRed()**, **getGreen()** and **getBlue()** which **return** an **int** between 0 and 255 indicating how much of that primary color is in the **Color** object. |

|  |  |
| --- | --- |
| **04a** | **Examine the Starting Files**  Load **Lab11Tester.java** in the edit window, like **Figure 07**.  The **main** method only has a few statements to begin with.  The **main** method is responsible for setting up a grid of random red cells.  **Figure 07**    **In this open space, you need to insert the code to accomplish “Objective #1”.**  ***See steps 02a and 02b.*** |

|  |  |  |
| --- | --- | --- |
| **04b** | **Examine the Starting Files**  Load **SortCritter.java** in the edit window, like **Figure 08**.  The **SortCritter** object sits in the (0,0) grid location.  This object creates an **ArrayList** of the **Location**s of all the **Cell** objects.  With each *Step*, the **SortCritter** places the brightest **Cell** at the end.  **Figure 08** | The **getOccupiedLocations** method returns an **ArrayList** of **Location** objects.  These will be all of the objects in the **Grid** starting at row 0, column 0 and ending at row 21, column 34.  Even though the **Grid** is  *2-Dimensional*, the **ArrayList** of **Location** objects is  *1-Dimensional.*  Index 34 stores what is at row 0, column 34. Index 35 stores what is at row 1, column 0. It is like it *wraps* around.  Remember, index 0 is where the **SortCritter** object is stored. You need to start your sorting at index 1. |

|  |  |  |
| --- | --- | --- |
| **04c** | **Examine the Starting Files**  Load **Cell.java** in the edit window, like **Figure 09**.  **Figure 09** | This file is ready to go!  When a **Cell** object is constructed, it will automatically have a color that is a random shade of red.  **DO NOT CHANGE**  **THIS FILE!**  The **act** method is supposed to be empty. Like a **Rock** we want a **Cell** to do nothing. |
| **05** | **Review the Objectives**  There are two distinct objectives for the students.  1. You need to create the grid with random red cells.  This is done in the **main** method.  The grid is 22 X 35.  Each **Location** contains a **Cell** object, which is like a square **Rock**.  The **SortCritter** object is in the (0,0) location.  2. You need to complete the **act** method of the **SortCritter** class.  **Critters** act on other **Actor** objects.  The **SortCritter** does not move. It only manipulates other objects.  **Critter** objects acquire an array of objects to manipulate.  This time the array contains all the **Cell** objects.  The job of the **SortCritter** is to place the brightest cell in the correct place.  After 768 steps the grid is sorted. |  |

|  |  |  |
| --- | --- | --- |
| **06** | **Graded Lab Assignment**  For **100 Points** write the program so that it displays the result of each comparison pass. In other words, each time that **Step** is clicked a complete comparison pass is completed and one object is in the correct location at the end of the grid.  For **110 Points** also write the program so that it displays the result of each individual comparison. If you wish, you can speed up execution if you change the number of rows and columns to 10. |  |