|  |  |
| --- | --- |
| **AP Computer Science GridWorld Case Study (GWCS)** | |
| **GridWorldLab07** | **The Grid and Location Classes**  **Creating a RandomActor Class** |

|  |
| --- |
| **Lab Objectives** |
| Understand how classes can contain other class objects with composition.  Understand the interaction of the Grid class and the Location class.  Introduction to the methods of the Grid class and the Location class.  Create a RandomActor class. | |

|  |
| --- |
| **Lab Prerequisites** |
| **Completed ExpoJava, Chapter 11, and completed GridWorldLab06**  Understand inheritance and composition.  The 110 point version also requires understanding of 2D Arrays. | |

|  |  |
| --- | --- |
| **Lab Sequence of Steps** | |
| **#** | **Actions** | | **Comments** |
| **01a** | **Getting Started**  Create a **GridWorldLab07** project.  Load the **RandomActorRunner.java** in the edit window.  The result is shown by **Figure 1**.  **Figure 1** | | The **RandomActorRunner.java** file shows a standard **Runner** type **main** method, which construct a single **RandomActor** object and place the object in the **world** object. |

|  |  |  |
| --- | --- | --- |
| **01b** | **Getting Started**  Compile and execute the **GridWorldLab07** project.  Click the **Run** button. The result is shown by **Figure 2**.  **Figure 2** | The execution of the **RandomActor** object is anything but random. The blue object sits still in one location and does nothing. An **Actor** object is supposed to constantly reverse direction. This object is very dull. |
| **01c** | **Getting Started**  Load the **RandomActor.java** file in the edit window.  The result is shown by **Figure 3**.  **Figure 3** | The **RandomActor.java** class looks quite peculiar. There are various methods that have no program statements in the method body. Can you explain the execution behavior of the **RandomActor** object based on what you observe in the edit window? |
| **02a** | **Create the RandomActor Class**  You must complete the **RandomActor** class with the following behavior:  1. **Actor** objects are constructed with a random color.  2. With each step the **Actor** object moves to a new random location.  3. A **Rock** object is placed in the previous **Actor** objectlocation.  4. The **Rock** object is constructed a random color.  5. The **Actor** object is removed if a **Rock** object is in the new location.  *Essentially, if an Actor lands on a Rock, it dies.* |  |
| **02b** | **Create the RandomActor Class**  Figures 4 and 5 shows a sample execution of the completed **RandomActor** class.  **Figure 4 – during execution,** *Actor object is still visible.*    **Figure 5 – execution completed,** *Actor object is gone.* |  |
| **02c** | **Create the RandomActor Class Lab**  The lab assignment has four different grade versions:  80-Points: The **Actor** object moves to random locations, but does not deposit any rocks.  90-Points: The **Actor** object moves to random locations and deposits rocks of random colors.  100-Points: The **Actor** object moves to random locations, deposits rocks of random color, and removes itself from the grid if the next move is occupied by a rock.  110-Points: *Can only be done if the 100-point version has already been completed!*  Just like the 100-point version except the **Actor** object needs to visit every cell in the **Grid**. This is done by creating a 2D array of **boolean** to keep track of all of the cells that have been **visited** so far. If the next random **Location** is a **Location** you have already **visited**, a new random **Location** needs to be chosen. After the **Actor** object has placed **99** rocks in the **Grid**, it can stop and does not remove itself. It is true that this could be accomplished using various methods of the GridWorld classes, but to get credit for the 110-point version, you need to use a 2D array of **boolean**. *Now, the Actor has the ability to “look before he leaps”.*  **Sample output at the conclusion of the 110-Point Version** | |