Design of Simple PacMan Program in Java:

Techniques Used and Discussion of Future Enhancements

Candace Holcombe-Volke

Johns Hopkins University

Design of Simple PacMan Program in Java: Techniques Used and Discussion of Future Enhancements

**Discussion**

**Flow of Program Execution**

**Game initiation and game board display.** Progression through the PacMan game depends on initial player input to build the game board to the desired size, and continuous player input to determine PacMan’s action. User input is collected through use of the Scanner utility. The main method initiates play by prompting the player for the height and width of the board, then invoking the createDisplay method to build and display the initial game board as a two-dimensional array, with size of height by width. createDisplay uses loop control variables to loop through each element in the array assigning a String (“.”) to each, which will be reflected as unvisited spaces on the board.

**Maintaining the game board and generating random cookies.** After assigning each element through the loop, createDisplay uses Math.random to randomly generate integers between 0 and the width, and 0 and the height of the array (game board grid) and assigns Strings (cookies, or “O”s) in the array elements using the generated numbers as array indexes. createDisplay checks whether the value of the random element has already been assigned a cookie, and if it has, it repeats the process one additional time ensure 15% of the elements are “O”s. It repeats this process the number of times which is equal to 15% of the total number of elements in the display array. Just before printing the array, createDisplay reassigns the starting position with the currentDirection String variable, and loops through each element again, this time printing the board. Once complete, the initial game board is returned to main as a two-dimensional String array called display, and assigned to the two-dimensional String array called grid. grid stores all updates to the game board throughout game play. movePac is the method responsible for displaying grid array by looping through each element of the array and printing the value.

**Rotating and moving PacMan.** The menuSelection method collects the user input and determines the flow through of the rest of the program. After it is executed, main passes the menuChoice (the return of menuSelection) into a while loop, where control remains during the remainder of the program’s execution.

Switch cases 1, 2, and 3 of the while loop in the main method are responsible for the action of the game, including turning PacMan clockwise or counterclockwise, moving PacMan to a new space, and storing and displaying the updated grid and move statistics. main method uses methods turnLeft, turnRight, and movePac to maintain and update PacMan’s direction. The turning methods are passed the currentDirection argument, which is stored in main, and then, through use of an if statement based on the value of the currentDirection at invocation, reassigns the new direction to currentDirection and returns the update to main. movePac then then updates the currentDirection of the current location of PacMan (stored as indexes of grid in variables int s, and int t) and re-displays the grid. The process is similar when PacMan moves to a new space, however checkOkToMove is first invoked to ensure that PacMan is not attempting to go outside the bounds of the grid, by validating that the new location exists in the grid array – that the new index element will remain greater than zero.

**Alternative Approaches**

One aspect of the program that could have been designed differently is the way the current direction is maintained and reassigned. In this program, the value of the currentDirection variable is used to determine the direction of PacMan. Alternatively, the value of grid[s][t] (PacMan’s current location) could be evaluated for the value of the String of that element. This would eliminate the need to create the currentDirection variable, however would require passing the whole grid array as well as the s and t integer variables into each method that uses that information.

This program does not use, to a large extent, nesting methods within methods. One example of this is the separate use of menuDisplay and menuSelection methods. The menuSelection method always follows the menuDisplay, however menuSelection is not invoked at the end of menuDisplay, but rather is invoked from main immediately *after* menuDisplay. The positive side of this approach is that it allows more flexibility to reuse methods without unnecessary or unintended outputs. One the negative side, it results in more disjointed code.

Another element that could have been done differently is the placement of the control of game play. Currently, the main method has a lot of responsibility for the action of the game. Another method could have been created that separates the duty of determining the menuChoice and directing the subsequent methods to perform their functions.

**Lessons Learned**

In working on this project, I started out quite uncomfortable two major elements that were required to complete the program: use of arrays, and passing and returning arguments between methods. As I worked through the project, I became more comfortable with these elements. However, the framework for the program is built on passing arguments from main and returning to main almost exclusively. As I develop my skills, I expect to become more comfortable understanding the scope of variables and having more ability to plan ahead for which method will require use of each variable.

I also created a number of unnecessary variables, such as currentDirection, on which my program relies heavily. I now understand better how the returns can be used in place of declaring specific variables.

As I progressed through the program, the main method grew in complexity, more than I was originally anticipating. I found through this project that breaking down the steps of the program into smaller “pieces” might help more appropriately determine the methods that will be needed.

**References**

*Creating and Using Java Arrays of Primitive Types* [PDF document]. Retrieved from https://blackboard.jhu.edu/bbcswebdav/pid-4598052-dt-content-rid-16762509\_2/xid-16762509\_2

*Defining Class Methods* [PDF document]. Retrieved from https://blackboard.jhu.edu/bbcswebdav/pid-4598038-dt-content-rid-16762662\_2/courses/EN.605.201.82.SP17/Module%2003/Module%204A%20Slides.pdf

*Method Documentation* [PDF document]. Retrieved from https://blackboard.jhu.edu/bbcswebdav/pid-4598044-dt-content-rid-16762668\_2/courses/EN.605.201.82.SP17/Module%2003/Module%204G%20Slides.pdf

*Passing Arguments to Methods* [PDF document]. Retrieved from https://blackboard.jhu.edu/bbcswebdav/pid-4598040-dt-content-rid-16762664\_2/courses/EN.605.201.82.SP17/Module%2003/Module%204C%20Slides.pdf

*Returning Values from Methods* [PDF document]. Retrieved from https://blackboard.jhu.edu/webapps/blackboard/content/listContent.jsp?course\_id=\_144009\_1&content\_id=\_4598034\_1

Schildt, H. (2014). *The Complete Reference* (9th ed.)*.* New York, NY: McGraw Hill Education.

*Some Built-In Java Class Methods* [PDF document]. Retrieved from https://blackboard.jhu.edu/bbcswebdav/pid-4598043-dt-content-rid-16762667\_2/courses/EN.605.201.82.SP17/Module%2003/Module%204F%20Slides.pdf

*The Scope of Method Arguments & Variables* [PDF document]. Retrieved from https://blackboard.jhu.edu/bbcswebdav/pid-4598039-dt-content-rid-16762663\_2/courses/EN.605.201.82.SP17/Module%2003/Module%204B%20Slides.pdf

**References**

*Using Arrays and Strings with Loops* [PDF document]. Retrieved from https://blackboard.jhu.edu/bbcswebdav/pid-4598054-dt-content-rid-16762688\_2/courses/EN.605.201.82.SP17/Module%2005/Module%205C%20Slides.pdf