Analysis of PacMan Game Program

Design Decisions and Reflections

by: Alina Grigorovitch

Introduction to Programming Using Java, section 82

Program Design and Analysis

The PacMan Game design incorporated a main method and additional methods that were called either within the main method directly or inside of another method. The main method divided the code into major sections in the following order: variables, game setup, and gameplay. Following the end of the main method were the supporting methods used in game setup and gameplay.

The variables section contained the class variables x and y, which were the rows and columns of the grid array, respectively, and which were entered by the user. Additional variables were those required to set up the initial grid to be passed to the gameplay: the char double array startGrid, the Pacman character, the cookie character, and the number of cookies, numCookies, determined by x and y. The variables were all used to then construct the initial grid using the following methods in the following order:

1. fillGrid, which used a for loop to fill each space in the array with a '.' character.
2. addCookies, which used the built in Math class to generate a random number between 0 and 1 multiplied by twenty percent of the total number of spaces on the grid.
3. addPacman, which initialized char pacman at the 0,0 position of the grid.

These functions were separate methods for the sake of clarity. No design issues encountered here as these methods used global variables. This created an initial grid but did not print it as printing the grid was a method of the gameplay section. The section finished by printing the display menu before the first round of gameplay.

An additional group of variables was then initialized below, separate from the array variables for the sake of clarity. These were a new char double array called grid, which was initialized to startGrid, the int counters for #moves and #cookies, and the int command, initialized to zero.

Following was the major design structure of the PacMan Game: a do...while loop that iterated through each step of a single play of using “while command != 4”. It began with calling the printGrid method which received the array grid, and prompting for the user command. A switch loop then processed the passed command variable and called the appropriate method based on the input.

Case 0 was simple and called the printMenu method again, which had previously been called during game setup.

The turn methods for cases 1 and 2 returned the new char of the pacman, thereby keeping the status of the pacman. turnLeft contained another switch loop that iterated through the four possible conformations of the pacman variable. Each case then searched the entire grid for the pacman position using a double for loop and, when found, changed that character to the appropriate character for the move. The turnRight method mirrored that design.

In case 3, the boolean method checkCookie was called to check whether the next position, based

on a switch loop iterating through the possible pacman characters, and incremented the variable eated if it returned true. This method used the exact same structure as the next method which will be described. Control then returned to the switch loop in the main method and called the move method which contained a double for loop to again iterate through the array and find the pacman, then pass the coordinates of that position a switch loop to determine the correct move direction based on the pacman's char status. Each case called the clearPath method to place a space character in the current coordinates and place the relevant pacman character in the coordinates of the next move.

The last case, 4, called the quit method, which printed the statistics. Since this command met the requirement of the do..while loop, control exited the loop and the game was over.

My initial thought for keeping track of the pacman's position was to call a currentPosition method at the beginning of each do..while loop and return an int array of size two, which could then be passed to the command methods for turnLeft, turnRight, and move. However, this gave me a java.lang.NullPointerException error, and I employed the more straightforward but less elegant method described earlier, finding the pacman's position only at the moment when necessary.

My initial idea for the move method was also quite different. I had planned to incorporate calls to additional methods to check for the cookie, inside the move method, an clear the path, but this gave me a java.lang.NullPointerException error, and made tallying moves and cookies a bigger hassle, as explained earlier above.

Another idea had been to make the gameplay loop a separate method with the do..while loop inside that, and pass all of the necessary variables to it, and then pass those variables to an enactCommand method containing the switch loop that would iterate through command input choices. However, this was a poor design as the required variables could not be returned outside of the method and was unnecessary. I thought having the do..while loop in the main method was satisfactory and not too messy.

Given more time, I would have attempted to fix the java.lang.NullPointerException error with exception handling to keep the more elegant version of the code for the command methods. Second, I would have found a way to implement the method to find the pacman's position instead of copying that code into each method that required it, for example by adding it in the do..while loop and passing the variable to the command methods. I would also have added a loop in the addPacman method to check that, if the pacman had been set on a cookie, the new cookie added did not also happen to fall on a cookie, ad infinitum. Finally, I would also have added more enhancements, such as an super cookie worth 10 points that would appear on a random spot once a certain percentage of the cookies had been eaten and move by randomly generated commands.

In sum, looking back there are many different approaches that I could have taken to the design which would have been better choices for the long run, but which would have required more time to plan and debug.

References

1. “Java: The Complete Reference 9th Edition.” Schildt, Herbert. Oracle Press, 2014 McGraw Hill.
2. “How to solve the java.lang.NullPointerException error?”, stackoverflow.com. http://stackoverflow.com/questions/10464547/how-to-solve-java-lang-nullpointerexception-error