Front-End Specifications

Last Modified: 09/12/2011

Specifications

Once you've fully developed the back-end, create a class to model a Monopoly game and contain the user interaction, according to the specifications given below. The specifications are given in the order you should complete them.

Defining the Board and State

Begin by meeting the following requirements to get your game class set up and see your board location class hierarchy in action too:

1. This class must have an array to hold the game board and an array to hold the players.
2. It must hold the *state* of the game, so in addition to the information contained within these objects, it should know the current player and the current dice value.
3. Provide a constructor that sets up an instance of the game. (It isn't necessary to take in values of every data member as parameters. If something doesn't change from game to game, hardcode it.) (You'll likely add to this later -- only part of it needs to be done to test as in #6.)
4. Have a method that defines all information about the 40 locations that make up the game board array. This method may simply contain hardcoded method calls. This should set up what doesn't change from game to game, e.g. the names of the spaces on the board, costs, rents, costs of houses. You may use official game information or may opt to use an alternative board setup, e.g. Penn Stateopoly.
5. Provide a method that prints out all information about all spaces on the board in order. What information you display about each space depends upon what type of space it is (but this method should be very simple if you've done the OOP right).
6. Create a main program that creates an instance of the game and tests that the population and printing of the board is working correctly before proceeding. (Take this out in your final draft.)

Game Setup

Here are a few notes on getting the game set up before play begins:

7. You don't need to implement user interaction to set up who's playing the game. It is acceptable to use constants to define the number of players and their tokens initially. It is just as acceptable to hardcode this information in your main program as pass it to your constructor or a modifier that helps with setup. Please use 2 or 3 players for the version you show in your demo.
8. Use random number generation to pick who goes first. (Where should this appropriately go? Think about that.)
9. Provide a method that sets up a "faster game"/"demo mode": In this mode, set up a configuration where you've sold to each player a few properties and you can use it to test and demonstrate game play in action without having to make a few trips around the board before someone has to pay rent. Call this in your main method before kicking off the game so we could comment the call out if we wanted to start a full game instead.
10. Incorporate this into your test driver from #6. (Again, remove testing in your final draft.)

Game Play

Here are the requirements for how game play should be implemented with user interaction:

11. Use random number generation for rolling the dice.
12. Have a continuously-running menu that lets each player take his/her turn. (Do some planning before coding!) Follow these specifications:
    - Precede each turn by displaying where all players are on the board, how much money each player has, and a list of each players' properties. (List the names of the properties only; do not give their full state. This status listing is to be concise.)
    - Begin each turn by giving the options to manage houses/hotels, roll the dice, view all details of the board, or end the game.
    - Upon dice rolls, respond to the spaces where the player lands with additional options that are specific only to the space where the player landed. (Note that the Food for Thought and Programming Aids page contains a function you can use to respond to menus; you should use it instead of making new code.) (Hint: check out the `String` class in the Java API for help in dealing with the results.)
    - Don't allow the user to do anything that violates the rules of the game. If your back-end logic is well-designed, you shouldn't have to do anything in the front end to meet this specification.
    - Have a clear submenu and clean user interaction for managing houses and hotels. Users should have the option to buy and sell houses and hotels and only be given options in this submenu that make logical sense.
    - Handle doubles, i.e. when a player rolls the dice and gets the same value on both dice, he/she gets an extra turn. (You do *not* have to handle the standard rule that sends a player to Jail after rolling doubles three times, as we're not implementing jail.)
    - Handle giving each player $200 for passing "GO."
13. The implementation of game play must be broken up into multiple sensible methods.
14. When a player ends the game, display all details about the state of the game. (No need to determine a winner.)

Tasks and Deliverables

Task #4: Start Game Class

Timeline: *Thursday, 9/22*

Start your class to model the game accordingly to the specifications above. Complete requirements under **Defining the Board and State**. No deliverable is required.

Task #5: Game Setup / Deliverable #3

Timeline: *Friday, 9/23, through Saturday, 9/24 (The deliverable isn't due until Monday because of the weekend, but you should really be deep into the programming of the game play over the weekend.)*

Complete requirements under **Game Setup**.

Requirements for Deliverable #3 are as follows:

- Prepare a report with code for your Monopoly game class meeting only requirements #1-#10 above, including the testing, and a sample run showing the test driver working.
- Turn in this report to Doug's office **before 1 p.m. on Monday, 9/25**.

Task #6: Complete the Game

Timeline: *Saturday, 9/24, through Tuesday, 9/27*

Complete the remainder of the game class/UI requirements.

Test your complete project carefully. Review your code carefully, correcting any errors you find and optimizing it. Leave yourself Wednesday to prepare your report and proofread.