Deadwood

Project Analysis

By Mason Heaman

Cohesion

Functional - Setview holds functions that are essential to the responsibility of updating the rooms of the board. The class does not tackle any additional responsibility and is therefore intuitive to change when the setView display wants to be altered.

Procedural - The startGame endDay and endGame functions are grouped in GameModel because they will always execute in a particular sequence: startGame, endDay*n, endGame.

Temporal - The card and board XML Parser functions are grouped because they will be processed before the initialization of the board UI.

Logical - The functions in the controller take on many tasks, but they all hold the logical goal of taking input from the user and relaying it to the necessary module.

Coupling

Data - The most common type of coupling seen in our project due to the use of the Observer Pattern. Two primitive data types, namely a data string and update type were passed from notifyObserver to update.

External - The data held in the board and card XML files is used throughout the entire game in classes such as Parser, Room, SceneView, etc. A change to this external data source will affect the game as a whole.

Common - The use of enums throughout the program made for easy updating, but it does result in lots of global data sharing.

Control - The use of enums in notify and update methods also resulted in control coupling. Passing a certain enum to one of these methods will control their internal logic.

SOLID Design Principles

Single Responsibility Principle - The private classes RoleChoiceListener and UpgradeChoiceListener have the single responsibility of sending data denoting the action button pressed.

Open-Closed Principle - exemplified in the Money class. This class can be extended to allow for different types of money to be used in the game. The two extensions present in the initial version of the game are Credits and Dollars.

Liskov Substitution Principle - Not Utilized
Interface Segregation Principle - Not Utilized
The Dependency Inversion Principle - The Room class is used as an abstraction all throughout the program so that high-level modules did not depend on the low-level SceneModel and CastingOffice modules.

Project State

Our project is currently in a working state with a few bugs. The main bug we see is that sometimes duplicates of player dice will appear after a scene wraps. We believe that this is due to a combination of the dice drawing and scene wrapping functions but have not had sufficient time to explore the cause. Given more time we would step through the interaction of these two functions approaching a scene wrap and look for unnecessary dice creation.

Assumptions

While coordinates were given for objects such as upgrades and roles, it was not described in the specifications that these areas would be clicked by the user to enact an upgrade. We made the assumption that a player will make action decisions by clicking buttons on the right panel. We found this to be more intuitive and descriptive to the user, while also making implementation smoother.

Testing

The game was played multiple times with different numbers of players.To allow bugs to be found efficiently, error messages were printed to the standard error at key points in the program (Scene logic, Observer methods, etc). The IntelliJ debugger was then used to step through the program at the reported location and diagnose the unexpected behavior. A “next day” button was implemented in testing to allow us to efficiently step through days and test that end of day and end of game logic was performing correctly. We ensured that scene logic was carried out correctly and that location, money, and rank were updated correctly. We also ensured that the game determines the correct winner and sends this value to the command line.

Challenges

The most difficult part of this assignment was iteratively changing the design of our project as we saw new requirements arise. I would have preferred to tackle the Deadwood assignment in an iterative process where design and requirements could be better assessed as the project went on. We ran into many hardships that would have been solved with a better understanding of requirements and module interactions that an iterative process would provide.