Course: CS 246

Assignment 5 - Group Project

Group Member: Isobelle Wang, Zhige Chen, Lynn Li

Project: Watopoly

Plan of Attack

Questions

1. Would the Observer Pattern be a good pattern to use when implementing a game board? Why?

The observer pattern is well-suited for this game because it allows the Player and Building classes to inherit from the abstract Observer class, and the Building class to inherit from the abstract Subject class. First of all, this enables ownable properties to notify players who own buildings, as well as other ownable buildings of the same type. This is useful for determining whether buildings can be improved or traded to other players, and also notifies all other buildings (observers) in its monopoly. Secondly, the OwnableBuilding class is a subject that notifies its observers when its status changes or certain events occur. Similarly, the UnownableBuilding class is a child of both the Observer and Subject classes, which allows players to know the status and number of active Roll Up the Rim cups in real-time. This enables each player to make informed decisions about their choices, as the total number of cups owned by all players cannot exceed four.

2. Suppose that we wanted to model SLC and Needles Hall more closely to Chance and Community Chest cards. Is there a suitable design pattern you could use? How would you use it?

We believe that the decorator pattern will make SLC and Needles Hall more flexible, as it allows for the creation of both actual and additional models, such as the Roll Up the Rim Cup. When landing on either of these buildings, there is a rare chance (1%) of receiving a winning Roll Up the Rim cup, which allows the player to bypass the DC Tims line for free. However, at any given time, there cannot be more than four active cups (meaning that if four cups are active, there is a 0% chance of receiving another one). To handle these events, we would like to implement the decorator pattern. This allows the program to update the status of events as needed and to check the next event if the previous one does not apply. Additionally, this approach can facilitate the addition of

new events in the future by implementing them as decorators, which will decrease the modification needed and improve the readability of the program.

3. Is the Decorator Pattern a good pattern to use when implementing Improvements? Why or why not?

The use of decorator pattern for the implementation of improvements is not recommended due to the high complexity of the different situations that can arise. Although using decorator pattern would add flexibility to our program and adding functionality would be easier in the case, there are about 132 integers related to the tuition with improvement, which makes it difficult to implement using the decorator pattern. Furthermore, some ownable buildings in the same monopoly may have different tuition amounts with the same level of improvements, making it harder to build the data with the decorator pattern. If we want to use decorator pattern, we would need to implement 5 concrete decorator classes, each for 1 level of improvement, this would be tedious and unnecessary since we can implement this through if statements and methods. In summary, building a whole design pattern for this simple feature is not worth the effort.

Project Breakdown and Specified Due Dates

Task	Asignee	Due Date
1. UML creation	Isobelle Wang (y3998wan)	Mar 30, 2023
2. Answering questions	Group discussion	Mar 30, 2023
3. Plan of attack	Lynn Li (j2557li)	Mar 30, 2023
4. Header files creation	Isobelle Wang (y3998wan)	Mar 31, 2023
5. Test suite (Black box testing) creation	Zhige Chen (z738chen)	Mar 31, 2023
6. Command interpreter Controller class implementation	Lynn Li (j2557li)	Apr 2, 2023
7. Main function implementation	Zhige Chen (z738chen)	Apr 2, 2023
8. Square class implementation	Lynn Li (j2557li)	Apr 2, 2023
9. Grid class implementation	Isobelle Wang (y3998wan)	Apr 2, 2023
10. Subject, observer, factory class implementation	Isobelle Wang (y3998wan)	Apr 2, 2023
11. Player class implementation	Zhige Chen (z738chen)	Apr 3, 2023
12. Building class implementation	Lynn Li (j2557li)	Apr 3, 2023
13. OwnableBuilding::AcademicBuilding class implementation	Isobelle Wang (y3998wan)	Apr 6, 2023
14. OwnableBuilding::Residence && OwnableBuilding::Gym implementation	Zhige Chen (z738chen)	Apr 6, 2023
15. UnownableBuilding class implementation	Lynn Li (j2557li)	Apr 6, 2023
16. Test suite (Grey & White box testing) & Debug	Group discussion	Apr 8, 2023
17. Additional fancy functionalities implementation (if possible)	Group discussion	Apr 10, 2023

Description of Tasks:

1. Draw the UML and get a basic understanding of the structure of this program, i.e the inheritance relationship between classes, which pattern would be suitable, and which public functions would be needed to implement this class.
2. Answer the questions given in the project description through group discussion.
3. Create a plan of attack to keep track of milestones and tasks to be finished which would help group members to get a better sense of the total amount of work to be done.

4. Create all header files with class declarations, public function signatures, and basic private fields, and assign the files to group members. At this point, abstract classes and pure virtual functions should be declared.
5. Create black box testing before actual implementation. Testing would include boundary/edge cases, corner cases, and other random tests based on intuition and experience/understanding of this game.
6. Implement the Controller class, which would enable initialize, load, save functionality. It would also enable switching to different modes (testing or enhancement) of the game.
7. Implement the main function, which would include accepting input from stdin, calling respective functions, and outputting to stdout or stderr.

8-15. Implement these respective classes, including implementing public functions to realize the functionalities as described, adding private fields or private methods if needed.

16. Add more tests to the test suite. At this point, each group member should write white box testings for the classes and methods she created as well as add grey box testings to the classes implemented by other group members.
17. If things go well, we should start coming up with new rules or functionalities for enhancement and implement those as much as possible. We would further discuss the tasks and related work distribution after completion of basic functionalities.