CS246 Final Project DD1 - plan.pdf

By: Daniyal, Giovani, and Helen

Before drafting the document, our group had our first group meeting via phone call, and we created our UML on call. The first thing our group worked on was architecting and drawing the UML. Doing this gives us a basic understanding and initial ideas regarding the constructor project. Our team plans to begin coding the project on Monday, July 18. We want to start off by creating all the necessary .h and .cc files regarding the UML we made. We want to ensure our dependencies and include files are set up correctly before starting to code our classes. Once this is done, we will create the template for the .h files and specifically define what each method does. Once that is done, we will work on coding the .cc files based on our earlier specifications from the .cc file. We plan to complete coding the main requirements by Friday, July 22. On Saturday and Sunday, i.e., July 23 and 24, our three will separately test the code, which would be helpful for us to find any possible bugs from different perspectives. If bugs are caught and cannot be fixed by the specific teammate, we will meet on Monday, July 25, to discuss and fix the bugs. Then, we will update the UML as our final version and discuss if there are any bonus marks we could possibly gain.

Additionally, since this is a large project, details (i.e., design logic and function documentation) we intend to write our thoughts and ideas in a shared document during the coding process. Eventually, this will be the design.pdf document which will combine our reflections. In order to make sure everyone in the team can receive the updated version of the code, we decided to use Github during the coding process. Once any of us starts coding, he or she will do git pull first. Similarly, once any of us changes the file, he or she will do a git push and post a message in the group chat to remind people that the specific files have been updated.

During our initial meeting, everyone picked our favourite part as the focusing responsibilities before DD2. We decided each team member will be responsible for a certain set of modules. Daniyal will mainly focus on Edge, Vertices and developing the Observer design pattern; Giovani will work on the builder module; and Helen will be responsible for Tiles, Dices and drawing the final UML. Since the Board module would be the central module with the most connections with other modules, we decided to maintain the board module together. We believe this will ensure that our code has high cohesion and low coupling. At the same time, as we mentioned above, we will make the design document while we are coding. For example, if Helen defines a void function titleInilization under the Tile module, she will update the design recipe and the requirements in the design file.

We plan to meet two hours per day starting on July 18th to communicate any issues we are facing, updates on the progress everyone has made, and features we intend to add to different modules. Once the coding is done, each of us will create ten test files to test the code thoroughly. Our goal is to ensure the code does not have any bugs and memory leaks under different situations; the code can successfully achieve all features mentioned in the project, and the code can be smart enough to run effectively. If possible, we will spend time making bonus marks.

All in all, we have created our design with the intent to maximize cohesion and minimize coupling. An example of this is our abstract Dice class, where we utilized inheritance to inherit the attribute of Dice for both the fair and loaded dies. Since a die will either only ever be fair or loaded, we have made Die an abstract class. Our board class is the main parent class which is responsible for most of the main methods to play our game. We use the observer pattern and text display class to show and update the game. We are concerned about potential segmentation faults since we will have a grid of tiles, and we want to make sure we don’t dereference a pointer not pointing to a tile (accessing a nullptr). Thus, we plan to test all edge cases and use GDB to debug our code if we find an issue. Also, we have to be careful with allocating memory and make sure to avoid memory leaks by clearly defining the instances where we have allocated memory from the heap in our code. At the same time, we will think about using smart pointers in the appropriate situation to avoid leaking memories.

Our first priority is to complete the main requirements. However, if all goes well, we do hope to add bonus features to our project. For example, we have discussed the possibility of creating a third die which would randomly be either a fair or loaded die depending on who is currently winning the game. This will make the games closer and more competitive. Another possible idea we have considered is using something similar to XWindow to make the output of the game look prettier and visually appealing.

Blow is our answers regarding the listed questions in constructor.pdf

1. You have to implement the ability to choose between randomly setting up the resources of the board and reading the resources used from a file at runtime. What design pattern could you use to implement this feature? Did you use this design pattern? Why or why not?

Using the Abstract Factory design pattern allows us to produce families of related objects without specifying their concrete classes. We need this pattern because we do not know the resources used until runtime, so we have to find a way to avoid specifying their concrete class. We can do this by first explicitly declaring interfaces for each distinct resource (e.g., brick). From here, we can have variants of the resources that follow those interfaces. This would help solve our problem of not knowing where the resources will be until runtime.

2. You must be able to switch between loaded and fair dice at run-time. What design pattern could you use to implement this feature? Did you use this design pattern? Why or why not?

We would use the Strategy design pattern to implement this feature. We know that loaded dice and fair dice are to generate a random number for the game to play, which means they share the same purpose. Strategy design pattern defines a family of dice, i.e., Dice, loaded Dice and Fair Dice, which can not only keep every Dice's encapsulation but also make the loaded and fair dice interchangeable.

3. We have defined the game of Constructor to have a specific board layout and size. Suppose we wanted to have different game modes (e.g. hexagonal tiles, a graphical display, different sized board for a different numbers of players). What design pattern would you consider using for all of these ideas?

We want to attach new behaviour for different game modes. We can do this using the Decorator design pattern. This will allow us to attach new behaviours to the game by placing these objects inside special wrapper objects that contain the behaviours. Polymorphism allows us to be flexible and change the rules while reusing our current code.

5. What design pattern would you use to allow the dynamic change of computer players, so that your game could support computer players that used different strategies, and were progressively more advanced/smarter/aggressive?

We want to dynamically change our computer players' strategy during runtime. The Visitor pattern allows us that you place the new behavior into a separate class called visitor instead of trying to integrate it into existing classes. The original object which performed the behavior is now passed to one of the visitor's methods as an argument, providing the method access to all necessary data contained within the object. This gives us the ability to separate the different computer strategies from the objects on which they operate.

6. Suppose we wanted to add a feature to change the tiles' production once the game has begun. For example, being able to improve a tile so that multiple types of resources can be obtained from the tile, or reduce the quantity of resources produced by the tile over time. What design pattern(s) could you use to facilitate this ability?

We want to attach new behaviour for our tiles during runtime. We can do this using the Decorator design pattern. This will allow us to attach new behaviors to the game by placing these objects inside special wrapper objects that contain the behaviors. Polymorphism allows us to be flexible and change the rules while reusing our current code. The decorator pattern will allow us to change these rules during runtime.