**Biquadris Initial Plan**

**By D. Zhang(d269zhan), M. Wang(m258Wang), Z. Chen(z473chen)**

**Plan:**

We plan to break down our projects into three main pieces: Commands (Controller), Subject, Cell, Block (model), Observer, Textdisplay (View), Graphicdisplay (View) (Based on our UML diagram). Each one of us will be responsible for one part. M. Wang will mainly work on the “Controller” part; D. Zhang will be responsible for the “Model” part and Z. Chen will focus on the “View”. As for the date, we decided to finish each of our own part before November 24, and combined our code into one whole project by November 27. The following week (November 27 – December 3) are used for debugging and adding new features.

**Answers to Questions:**

**Question 1 – How could you design your system (or modify your existing design) to allow for some generated blocks to disappear from the screen if not cleared before 10 more blocks have fallen? Could the generation of such blocks be easily confined to more advanced levels?**

A: We decided to create three types of classes: “Cell”, “Block”, “Board”. A Block is a composition of multiple Cells and so Block stores each of its components’ information. A Board stores the information of every Cell and current Block. So whenever a block needs to be deleted, the Board will locate this “block”, and then lookup its composition information to find the cells that need update. This will easily solve the problem no matter what level you are in Because Cell and Board will have a field tracking the current level you are in.

**Question 2 – How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?**

A: We plan to add a field in the Board class called level. This field is an integer indicating which level this board is currently in. Also, both the Cell and Block class also has this field. Beyond that, Board also has multiple Boolean fields such as isRandom, isHeavy, etc. This design is used due to the fact that there are only small changes between each level. So we can add some public methods according to these changes and once we know what level and what Boolean value returns true, we can call the related methods to achieve the request behavior of that specific level. This design considers code complexity and resilience to change. It’s a design we believe with minimum recompilation.

**Question 3 - How could you design your program to allow for multiple effects to be applied simultaneously? What if we invented more kinds of effects? Can you prevent your program from having one else-branch for every possible combination?**

To allow multiple effects to be applied simultaneously, we can add a vector of Booleans as a private field of the Board class. At position “i” of the vector, it represents whether the “Special Action i” is switched on or not. By doing so, we can add as much special action as we want and it is easy to implement and with minimum recompilation. When we actually try to execute these special actions, just simply loop through this vector and call the corresponding methods.

**Question 4 – How could you design your system to accommodate the addition of new command names, or changes to existing command names, with minimal changes to source and minimal recompilation? How difficult would it be to adapt your system to support a command whereby a user could rename existing commands (e.g. something like rename counterclockwise cc)? How might you support a “macro” language, which would allow you to give a name to a sequence of commands?**

We have a Command class as the controller. So when new commands are added, we just simply change the interface of Command and the Model part remain the same. This design is pretty easy for renaming the existing command since you only need to change the condition of the if-else statement, the body part is just calling the method and you don’t have to change anything. As for “macro” language, we can define some constant string containing the sequence of commands and use it as the command in the future.