CS246 FINAL PROJECT PLAN (Due Date 1)
BIQUADRIS

TEAM:
Hamza Bin Yusuff (hbyusuff)
Abrar Rakin (a2rakin)
A S M Taseen (ataseen)

Estimated Completion Date: August 5th

Project breakdown and sequence of action:
By now, all of us have gone through all the course content and have a clear understanding about the different components/sections of the biquadris. We will be using our UML and design ideas to write all the abstract superclasses by 31st August. After that, the next three days we plan to implement all the required concrete subclasses, stated functionalities, game logic, and the graphics. By the 3rd of August, we hope to be done with our basic program structure and then spend 1-2 days for thorough testing, any sort of debugging, and for possible enhancements. Throughout this period of eight days, starting from 29th August, we hope to stick to our plan and portray strong object oriented programming concepts and implementations through our version of the biquadris.

Division of work:

Abrar -
- Block
- Board
- Next Block

Hamza -
- Display
- Command-line interface

Taseen -
- Special Actions
- Command Interpreter
- Scoring

Question:
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?

Ans: We could use a “count” attribute in the abstract Block class to keep track of the number of blocks that will be generated after a block has dropped into the board and has not yet disappeared. If 10 more blocks have fallen and the block still exists, then we

In order for the generation of such blocks to be easily confined to more advanced levels, we could assign a level number to each block and only increase the count if blocks are created at that specified level.

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

Ans: The abstract Level superclass makes it very easy to introduce additional levels into the system. We can derive more concrete subclasses from the superclass, and make necessary additions to the subclasses as per the requirement of the new level. This ensures minimum recompilation as only the .cc file for the new level has to be recompiled.

Question:
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?

Ans: We plan to employ the decorator design pattern for allowing multiple effects to be applied simultaneously. We can use an abstract superclass to create the concrete subclasses for the effects and,using the decorator pattern, we can add effects independent of one another and so we would not need to have else-branches for all possible combinations of the effects. Moreover, we could derive more concrete subclasses from the abstract class and implement more effects by just adding the unique functionality of the effect onto and only on its derived subclass.

Question: 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? (We acknowledge, of course, that adding a new command probably means adding a new feature, which can mean adding a non-trivial amount of code.) 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? Keep in mind the effect that all of these features would have on the available shortcuts for existing command names.

Ans: We plan to use a ComManage class that has a vector of commands (strings). The vector stores all the commands as strings and new commands can be added by pushing the new command into the vector. We can add a public method for renaming any existing command. After that, the corresponding functionality will run when the updated command is called.