A5 Plan for Biqaudris Project

Group Members

Paranjay Agarwal (p38agarw), Daksh Prashar (d3prasha), and Jaskirat Pabla (j6pabla)

Project Breakdown

Date	Tasks to be Completed	Responsibilities of Group Members
Mar. 23	Reading over project specifications and choosing the project that we find most fascinating. Ensure that we understand how the game works and what is expected from us.	Paranjay, Daksh, and Jaskirat
Mar. 24	Working on the UML prior to coding anything and answering the questions on the Biquadris instructions document.	Paranjay, Daksh, and Jaskirat
Mar. 28	Working on Gameboard class.	We will split Gameboard's functions among ourselves.
Mar. 29	Cell, Block, and Score classes.	Paranjay: 1 Block class, Cell class, and Score class Daksh: 3 Block classes Jaskirat: 3 Block classes
Mar. 30	Implementing Level parent and subclasses.	Paranjay: Level 4 Daksh: Level 0 and 1 Jaskirat: Level 2 and 3
Mar. 31	Parent Classes: Special Actions, Board Sub Classes: Blind, Force, Heavy.	Paranjay: Blind Daksh: Force Jaskirat: Heavy
Apr. 1	Implementing Display parent class, Text and Graphic subclasses.	Paranjay, Daksh, Jaskirat
Apr. 4	Going through all work done so far and checking if everything works. Implement possible bonus features such as Level 5.	Paranjay, Daksh, Jaskirat
Apr. 5	Going over the entire project and implementing anything else we feel is necessary.	Paranjay, Daksh, Jaskirat

Group Answers to Biquadris Questions

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?

- To satisfy this condition, we can implement a new method in the *Block* class. This method can be called removeOld() and it will determine based on the current level of the game whether we should remove a block that has been there for more than 10 drops or not. To implement this function, we will add an additional vector of Block pointers, which will contain all the blocks that have already been played in the order of when they were played.
- Now, if the levels are not advanced once the size of this vector exceeds 10, then we will erase the element at index 0 of the vector. Along with that, we will also remove this block from the gameboard as well and shift all the blocks that are potentially above it downwards. This ensures that the size of our vector of block pointers never exceeds the length of 10, which also means that the maximum number of blocks that can be on the gameboard at one time is 10. Note, if the levels are advanced, then the vectors elements are not removed even if they exceed length 10, to confine this condition.

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

- We can accomplish this by implementing the factory design pattern. This is because, whenever we add a new level, we will simply add an additional subclass to the *Level* class. In this class, we will be able to override methods to cater to the needs of Level 5 or any additional level(s) the user wishes to add. By utilizing the factory design pattern, file dependencies are minimized, and low coupling will minimize recompilation.

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?

- We can accomplish this by implementing the Decorator Design Pattern. To add additional effects, we can create subclasses of the *Decorator* class which in turn will allow us to use these effects without an else-branch for every possible combination. This will be implemented in a similar way taught in class and used in a4q3b.

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.

- We create a commandSetter class that has methods for every new command and if one would like to add a new command, we would just add a new method to the class. Thus, only the commandSetter class will be recompiled without affecting the other classes. And if we want to change a command, we just change the if statement that checks for commands from the old command to the new command name.