Assignment 5 Due Date 1

Chengjie Huang (c267huan), Shida Yang (s368yang)

CS246 Nov 25, 2019

Plan of Attack

Milestone	**Requirements & Responsibilities**	**Estimated Completion Date:**
Be able to start Biquadris and is able to take in commands	The program is able to start and exit without complications. The game loop is able to start and the command interpreter is able take in commands. Valid commands will have no functions, however, they will print their intended purpose. **(Shida & Chengjie)**	Nov 25
Basic Game	Able to generate a block on the board. Level 0 will be implemented.**(Chengjie)** TextDisplay is able to print the board, level, score, and next block. **(Shida)**	Nov 24
In-game player interaction working	Able to move, and rotate blocks. TextDisplay is also able to output these movements. **(Shida & Chengjie)**	Nov 25
Get blocks to drop correctly	Blocks are able to drop and accumulate correctly on the board. **(Chengjie)**. Blocks when moved are not bounded by the boundaries and other blocks **(Shida)**.	Nov 26
Row Clearing and Scoring	Rows are automatically cleared when they are filled **(Shida)**. The score is updated after line clears **(Chengjie)**.	Nov 27
Other Levels and Block	Implement the remaining levels **(Shida)**. Implement the remaining blocks **(Chengjie)**.	Nov 27
Create Graphical Display	Create a graphical display for the game. **(Shida & Chenjie)**	Nov 28
Game without special features is done	All bugs and memory leaks have been identified and fixed. Features and classes not mentioned above have been implemented. **(Shida & Chengjie)**	Nov 29
Additional Features	Implement any additional features. **(Shida & Chengjie)**	Nov 30 - Dec 1
Update UML and Finish Report	Update the UML to match the code that has been written and write a report regarding the design of the code. **(Shida & Chengjie)**	Dec 2

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?

In our design, a Block is created by an instance of Level (an abstract class), which is owned by a Board.
We can allow for Blocks to be cleared from the screen if not cleared before 10 more blocks have fallen by modifying the Block class. In the Block class, we can add a field that tracks the number of blacks that have been dropped after this Block. Since the blocks are generated by an instance of Level, in the implementation of the generatedNextBlock() method by the concrete Level class, the Levels that are considered more advanced with generate a Block with the field initiated to 10; otherwise, it will be initialized to -1.
When the Block get dropped into place on the Board, the Boards will check if any rows have been completed to clear them by going through the board row by row. During this process, all the Blocks that are non-negative will have the tracking field be decreased by one and if the field becomes 0, the Block will be removed from the Board.
With this design, only the Blocks generated by the more advanced fields will self-destruct, and steps that are used to implement this feature already exist within the game such as having the Level generate the block and cycling through each block after each block is placed.

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

We designed the factory method to spawn blocks. If additional levels were to be introduced into the system, we can simply add new concrete level classes that inherit from the Level superclass. Methods in all other modules needs only minimal modification.

Keep in mind that the only class that relates on the Level class is the Board class. In order to have minimum recompilation, we can use a Level pointer as a private filed in board.h, and thus only declare the Level class in board.h instead of directly including the class. By doing so, modification of the Level class will not require the recompilation of other classes.

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

Currently we have three special actions: Heavy, Force, and Blind. If either Blind or Force are used, we will process these actions by directly applying them on corresponding objects (Blind on TextDisplay/GraphicDisplay, Force on nextBlock field of the Level class). We acknowledged

the possibility that multiple heavy effects may be applied simultaneously on a new block. Therefore we added a private integer field of “Heavy” within the Level class and Block class. Whenever a heavy command is added to a player, the “Heavy” field in the Level class will be incremented by one. When a player’s turn starts, a new block will be generated by the Level class as part of the factory method. The Level class will read the “Heavy” field, and create a block with the same “Heavy” field. The new block’s moves methods will ready from the “Heavy” field and decide what to do.

There are two advantages of implementing the special actions in this way.
a). This allows simultaneous uses of multiple actions without having one else-branch for every possible combination. Each special action is counted on their own.
b). If we invent more kinds of effects, we only need to create new counters for new actions. There is no need to modify previous actions.

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? (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.

To allow our game to easily accommodate the addition of new commands or changes to existing command names, with minimal changes to source, we will implement our command interpreter with an enumeration called Command that be easily added to and an if statement can determine which functionality to execute. By using an enumeration, we can prevent having to changes to every function call whenever a command name is changed. Also the compiler will output an error if we use an enumerated command that does not exist while it will not when we misspelled a string used in an if statement. To add a new command all we would have to do is add a new Command enumeration and an additional case for the if statement.

To support a command renaming system we would have a dictionary of commands using std::map<string, Command> where the key is the user input and the value is the Command enumeration.

In order to incorporate a macro language that would allow us to give a name to a sequence of commands we could simply change the previously mentioned implementation to std::map<string, vector<Command>>. Here a user can input one value and the command interpreter will execute multiple commands.