CS246-Biquadris-Plan of Attack

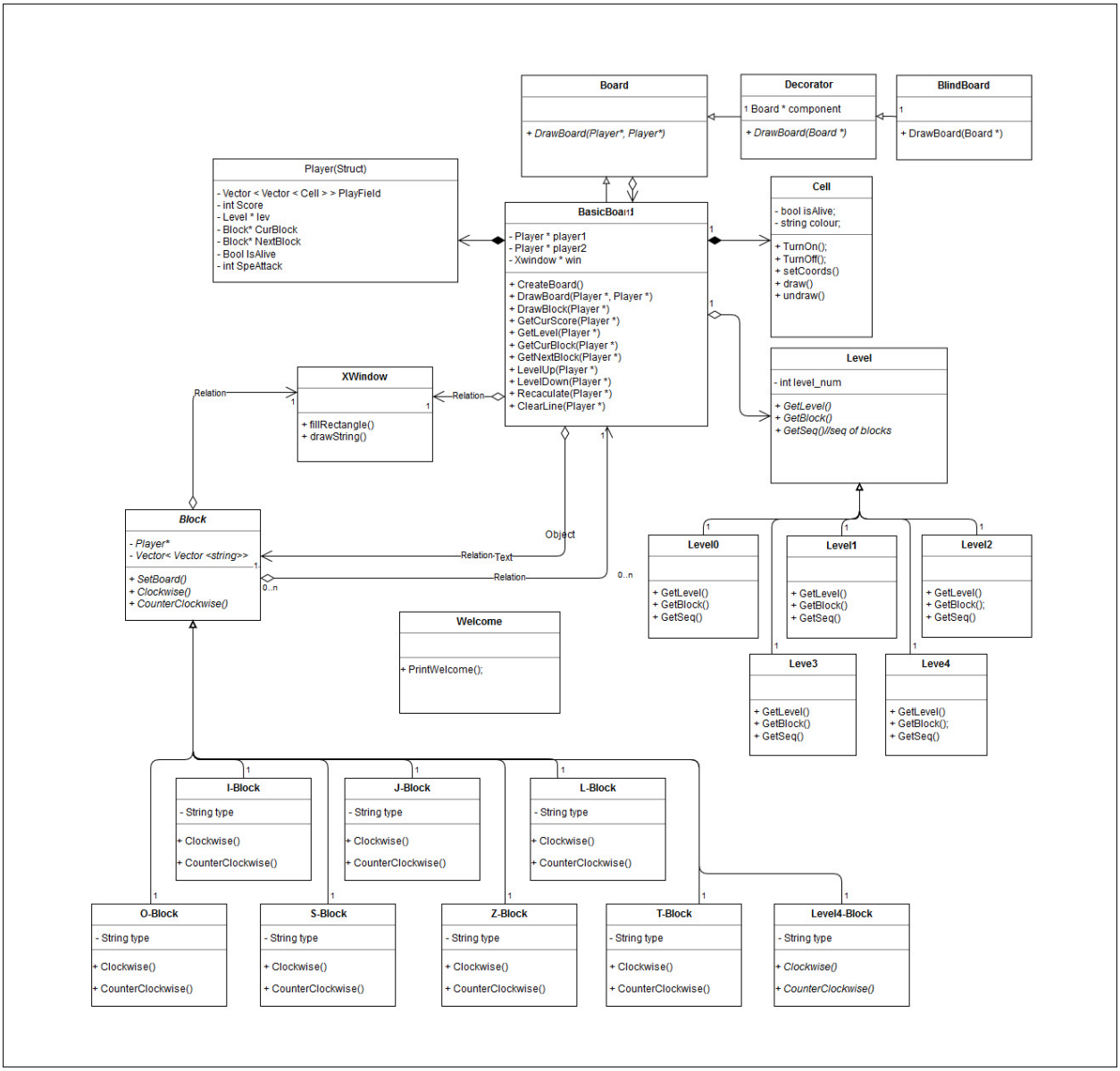
Yinyao Zhu (Amy)(y375zhu)
Yijian Wang (Harry)(y2549wan)
Jiadong Mai (John) (j4mai)

Date	**Content**	**Responsibility**
Nov 14	Going over the project, choose the one that we interest the most.	ALL MEMBERS
Nov 15	UML first look, brainstorm the possibilities. Design pattern options, and implementation thoughts.	ALL MEMBERS
Nov 16	UML first attempt (UML version 1.0). Plan of attack and distribution of responsibilities. Details of the required functionalities may change as the group codes.	Block: Harry Board: John Level and main: Amy
Nov 17	Group coding session NO.1 (3 hour coding session in DC or MC, 6PM to 9PM) Everyone will be present. Communication is required to make sure polymorphism works the right way within the classes we created.	Each person works on the part they are responsible for.
Nov 18	Group coding session NO.2 (3 hour coding session in DC or MC, 6PM to 9PM) Everyone will be present. Communication is required to make sure polymorphism works the right way within the classes we created.	Each person works on the part they are responsible for.
Nov 19	Check point 1 (3 hours): By now we should be able to perform the most basic actions on the game board. The display of the game will show up in text form as described in the assignment. ● Move the 7 types of blocks (left, right, down, drop) ● Have the blocks appropriately showing up on the board ● Basic level factory should be running appropriately: ○ Correct level should appear on the board. ○ Fstream functions should be working properly	Each group member presents what they have coded for their parts, includes a collective debug session if needed.

	○ "Next and current block display" functions in the desired way ● Ostream operator overloading for << should print/output the correct state for both Block and Board, as well as the corresponding level. ● Basic command interpreter is completed. Using these commands, blocks will perform basic movements on the board.	
Nov 20	Group coding session NO.3 (3 hour coding session in DC or MC, 6PM to 9PM) Everyone will be present. Communication is required to make sure polymorphism works the right way within the classes we created.	Each person works on the part they are responsible for.
Nov 21	Group coding session NO.4 (3 hour coding session in DC or MC, 6PM to 9PM) Everyone will be present. Communication is required to make sure polymorphism works the right way within the classes we created.	Each person works on the part they are responsible for.
Nov 22	Check point 2: At this point, the mainframe of the game is done. The basic functionalities of the game of Biquadris should act the way that is described in the assignment. ● Lines will disappear when they should ● No overlap or unwanted collision will happen ● Blocks shows up where they should ● Scores are recorded, basic counter is used. (sophisticated scoring is implemented later) ● Levels can be changed and are integrated with the board and blocks. Correct sequences of the blocks will be produced whenever needed.	Each group member presents what they have coded for their parts, includes a collective debug session if needed.
Nov 23	Special Action - heavy and force. By the end of this 3 hour session, the special actions of this game should be implemented correctly. Each group member is assigned with their special action that is the most related to their parts.	Force: Amy Blind and Heavy: Harry and John
Nov 24	Graphics By the end of this 3 hour session, the graphics of this game is completed using the Xwindow class. Tweaks and trick can be added if time allows the group members to do so.	ALL MEMBERS
Nov 25	Bonus features	Sound: Harry

	By the end of this 3 hour session, the bonus features of the game is created. Primary thoughts on bonus features: ● Sound is made when a row is eliminated ● Special decorator for “impressive” row eliminations (3 or more). ● Block colour and hopefully texture ● No delete ● Lives (if we have time)	Decorator: John Block Colour: Amy No delete: this function will be achieved by using only vectors in the program, through collective effort.

UML Version1.0



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 conned to more advanced levels?

We could add a counter for cell to count how many rounds a cell survives, so we could have record to check whether a cell need to disappear or not in count of how many rounds it survives. If in the advanced blocks there are rules about disappearing a cell in regardless of rounds, it will be easily connected to similar rules.

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

We can achieve this requirement by using Factory Method Pattern. For example, We can use factory method pattern on the *Level* class. In our *Level* class, there are 5 different levels as subclasses which are used to control the difficulty of the game. If we want to add one more level, such as Level 5, we can simply add a subclass (e.g. level 5) of the *Level* class, thus only the *Level* class will need to be recompiled.

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?

Decorator pattern will be ideal in this situation. Decorator pattern allows the concrete class to be decorated through its subclasses simultaneously. If we wish to apply multiple effects, we can simply add more decorators under the abstract class Decorator, and use the object to decorate the concrete class when it is required. Note that multiple decorators can be used at the same time (this is the essence of the Decorator pattern). In context of the special actions for instance, we can use the decorator pattern on the *Board* class. *BasicBoard* and *Decorator* class inherit from the *Board* class. *Decorator* class is an abstract class which only contains a pointer of the *Board* class. We add a *BlindBoard* class inheriting from the *Decorator* class. The *BlindBoard* has an effect that uses '?' char to cover some areas on the Board (details listed in the assignment). Again, if we wish to add more effects, such as board-frame or sound, it is easily achievable through the method of decorator.

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.

In order to make command names more flexible, for example, "lef" will have the same effect as "left", we will sort our command line in alphabetical order and decide which command will be run. We will store the commands using string, and compare each character with the commands we wish to have. For example, we will have "left" and "levelup" command all under (string[i] == 'l') catalog, then keep comparing until we can distinguish which command the client wants to run. Whenever we need to add or change command names to the program, we will go through our existing alphabetical order list and find the place wherever suits our new command.

In order to go through a sequence of commands(macro language), we could generate an array of shortcuts of command to go through the list of command names. For example, we can run a double loop. The outer loop is used to get the first keyword, the other one is used to execute one or more commands.

```
String command;
Vector <string> string_of_commands;

While (Cin >> command){
        If (command != one_of_our_commands){  // and is a sequence command key word
                //  we generate our multiple commands array here
                string_of_command.push_back(...);
        }
Else{
        String_of_command[0] = command; // eg.  ["drop"]
}
        For (int i = 0; i < string_of_commands.size(); ++i){
                If (...){// we find our single command key word here
                        // and takes action
                }
        }
}
```