UNIVERSITY OF WATERLOO

Faculty of Mathematics

CS246 Final Project Quadris

Final Report

Prepared by:

Mingze Lin

Shanpeng Wang

Jiayu Shao

Dec 2017

Introduction

This report is designated to reflect and conclude any progress my team has made during the completion of Project Quadris.

Overview (describe the overall structure of your project)

Overall, we tried the best to follow any rules to show the understanding of design pattern and style. In general, the code is designed to follow MVC, pattern, main file only reads from the command line (with help from class Command) and outputs to the standard output; it leaves the rest to the hand of class Board, while Board serves as the controller (controls the other classes).
We also adopted Observer pattern to deal with the relationship between Board and TextDisplay, GraphicDisplay. In order to generate blocks, we have abstract classes Block and NextGenerator, the former follows Factory Method Pattern, while the latter follows Template Method Pattern Lastly, we created class Record to keep track of any score-related information generated during gaming.

We use class Board to represent literally the “board”, there’s a Owns-A relationship between class Board and six other classes including Cell, Block, TextDisplay, GraphicDisplay, Record, Level0, Level1, Level2 and Level3. In addition, Board is a subclass of class Subject, as it is obervered by TextDisplay and GraphicDisplay.

In specific, firstly, A Cell represents a 1x1 square of Board. Every Cell has four fields named as row, col, state and kind. Row and col namely records the position of the Cell. State (is either 0 or 1) shows whether the Cell is being occupied or not and kind shows which kind of Block occupying the Cell. Cell has getters and setters to all the private field. In Board, we use a vector <vector <Cell *>> to store pointers of all the Cells on the Board.

Secondly, a Block represents one block of quadris. Each block has a vector of size 6 of Cell pointers. For all the block, 2 of the pointers will be nullptr, and four of them will keep track of the real location of each block. Such design allows us to maximize the reusability of codes. Block is a super-class of Bar, Square, Rect and Unique. Bar represents the I-block, Square represents the O-block, and Rect represents J, L, S and Z blocks since all of these four blocks can be contained in a 2x3 rectangle. As a special member, Unique represents a 1x1 block (used only for gaming level 4). Each time when count in Record reaches 5 in level 4, a Unique block will be called. In class Bar, Square and Rect, they each overrides rotate and crotate.

Thirdly, NextGenerator has a pure virtual method called generateNext(), it is the superclass of Level0, Level1, Level2 and Level3. When generateNext() is called, the block of corresponding level is created by each Level class and the pointer of the block will be returned. In Board, we store the pointer of Level-0, Level-1, Level-2 and Level-3.

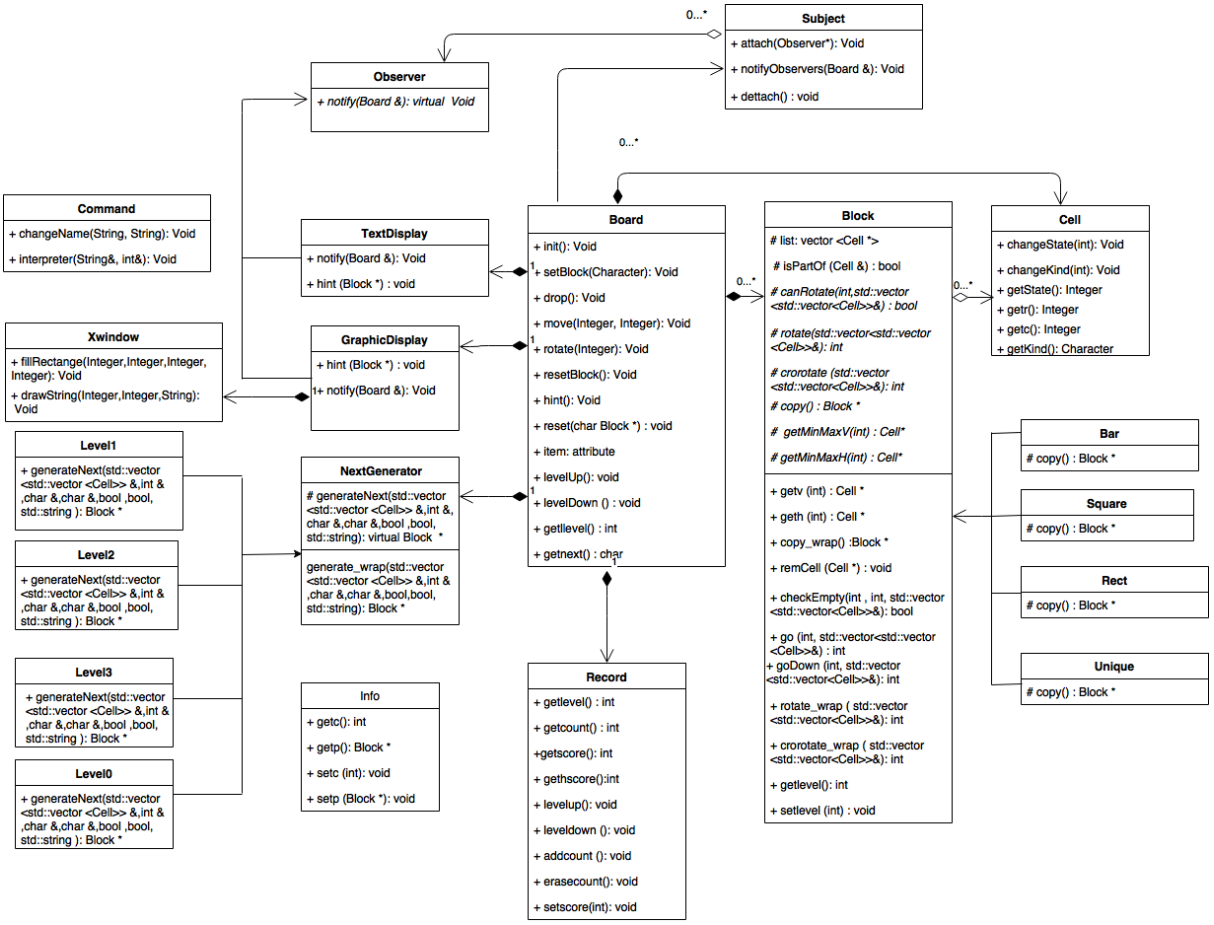
Moreover, TextDisplay stores a vector<vector<char>> to record all characters on Board and friends operator <<. The operator << can be used to print the whole Board by characters. The Board owns-a pointer of TextDisplay.

In addition, GraphicDisplay owns-a Xwindow object, and it will be notified by Board, whenever Board is changed, it can also access information of Board (i.e. level, score and etc.) by using the getters in Board. The Board owns-a pointer of GraphicDisplay.

Besides, class Record has four fields to record the current level, the current score, the highest score and how many blocks have been set up without one row being cleared. Record has getters and setter to access and modify fields. Record is stored in Board as a pointer.

Last but not the least, when the program is executed, one Board is initialized and one new Block is set up. We also have a Command class which stores all commands that our program can use. Main file passes all the input to Command class for interpretation on the “macro-language” and waiting for response.

Please refer to the UML for further knowledge and better understanding



Design (describe the specic techniques you used to solve the various design challenges in the project)

Indeed, Quadris is truly a challenging project, in order to achieve good style, we have tried our best to follow the single-responsibility rule, so that each class does one thing and one thing only, so each element mostly only perform one task. Also, we want the code to be reusable as much as possible, so we tried not to friend any redundant classes to any of our classes (note that only Board and TextDisplay need to be friended with standard ostream). In this way we achieved as much low coupling and high cohesion as we can.

Class Block and its subclasses(i.e. Rect, Square, Bar and Unique) are an example Template method pattern. In order to avoid hard coding as much as possible, method go and goDown is implemented in Block, it can be used by all subclasses. Method rotate and crotate are pure virtual, each subclass has overrides the method

in there own way. Notice that rotate and crotate are protected pure virtual methods and they are wrapped by public methods rotate_wrap and cotate_wrap. So that we followed NVI idiom and Template Method Pattern successfully.

As for class GenerateNext and it subclasses, they adopted the Factory Method Pattern to generate a Block pointer. Note that Method generateNext is protected and pure virtual, and is wrapped by generate_wrap, which is a public method. In this case NVI idiom is also well-preserved.

Moreover, we made TextDisplay and GraphicDisplay observers of Board, so that we can minimize the number of times to notify observers (which enhances the efficiency in the meanwhile). The relationship is based on the Observer Pattern.

Overall, as previously mentioned, we adopted MVC pattern, class Board serves as Controller, TextDisplay and GraphicDisplay serve as View, and the rest (i.e. Blocks, GenerateNext and etc.) serves as Model. In this way, when we encounter a problem, we can boil down the problem as soon as possible, and it also helps us to break down the project into three parts and assign to each group member.

Resilience to Change (describe how your design supports the possiblity of various changes to the program specication)

As we adopted several patterns in our coding, supposedly, it should be easy to add anything new to the code. For example, if there is some new type of Block, we can either make it one of the four already existed subclasses of Block, or we make it a new subclass of Block (make it part of the Template Method Pattern we have for GenerateNext and its subclasses). There is little change we need to make to the original code, as each file generally follows low coupling, it should be easy to reuse the code. (e.g. if there’s a new shape, it will “reuse” the methods from Block) Similarly, if we need to add a new level, we simply need to add it into the Factory Method pattern we have for GenerateNext and its subclasses.

In addition, due to the elegance of our very proud design and style, say if you would like to add a new display class which observes the game, we can simply make it a new subclass of Observer, and attach it to the observer list of class Board.

Last but not the least, say if you would like to add a new function to the game, due to the elegance of MVC pattern, it’s very easy to know where we need to make

changes to. (Say if you want the Block to be able to make a flip, then you probably should make changes to components in Model).

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?

Previously, we said that we would adopt observer pattern to monitor the drop of each block, and increment the count (a field of Board) each time; once the count reaches 10, a function will be called to remove the a block from the list of blocks. The previous idea of ours was absolutely correct, however, it seems to be redundant work to make it part of the observer pattern. Unlike GraphicDisplay and TextDisplay, we do not need it to observe every change made to the Board (e.g. rotate, move down and etc.), we only want it to monitor the motion of drop, and informs count to increment. Therefore, right now we believe that it suffices to simply modifies the method drop of Board class, once a block is dropped, method drop increments count by 1 and check whether count reaches 10 already, if so, it will call a specific method to remove a block from the list of block.

Could the generation of such blocks be easily confined to more advanced levels?

Previously, we thought that we can solve it by simple modifying the ways we overrides generateNext method in subclasses of GenerateNext class. However, we found it's even easier if we have a Record field in Board to simply keep track of the current level, and the drop method will increment count(a field of Board) if and only if we are at the "advanced level" and call specific method to remove a block from the list of blocks.

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

We would like to re-interpretate our previous answer to this question. Indeed the best way to minimize recompilation when introduce a new level is to use Template Method Pattern, because then we only need to add a subclass to abstract class GenerateNext, and overrides the generateNext method in the new class. And once we are done, we only need to recompile the newly added code, because we do not need to recompile any code that are not changed. (Using Template Method Pattern and Bridge Pattern allows us to keep as much code unchanges as possible)

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.

To accommodate the addition of new command names or changes to existing command names, we will create a command class that is strictly responsible for controlling all the commands. Specifically, we will be storing each command into two vector of strings in the Command class(One for the original command and one for the updated command) so when adding a new command we will just simply emplace_back a new string into our string vectors then implement its feature in the main, and when changing a new command we just need to change it based on the old string. In order to support rename, we will be comparing the input from user input and new updated vector of string in the command class and returning the original command name, so that main can still execute the propre command. When user wants to rename, they can simply call the rename function in Command class. To support macro language, when user typed a command, it will pass it to the Command Class to interpret it, in summary, use size to do a substring from the full name to match it, then pass the full version of the command back to main. We also improved our program so that it will check if no command has been matched or multiple commands have been matched, in this case, nothing will perform.

Extra Credit Features (what you did, why they were challenging, how you solved them|if necessary)

For our program, we deeply want to build a friendly environment(user interface) so that users do not need to type out the exact command to execute certain feature, instead a short version of it, in another word, macro language. In order to support macro language, we specifically build a Command Class to achieve this goal. When user typed a command, it will pass it to the Command Class to interpret it, in summary, use size to do a substring from the full name to match it, then pass the full version of the command back to main. We also improved our program so that it will check if no command has been matched or multiple commands have been matched, in these case, nothing will perform. One more thing, our program also supports rename, which can rename a certain command name to anything that user named.

What lessons did this project teach you about developing software in teams?

If you worked alone, what lessons did you learn about writing large programs?

We have learned a lot about developing software in team. Firstly, the use of design pattern can largely increase the efficiency of interpretation. For example, the observer pattern minimize the extra code we need to add if we want to add a new observer. For another example, Factory Method Pattern allows us to minimize the code to generate a new object with specific features. The most important thing of coding is not just about to complete the coding assignment, but to leave space for future maintenance (possibly will be done by coworkers, so the code needs to be clear in both structure and syntax to maximize readability). And considering working in team might meaning that the program will be very large, we should minimize recompilation of code by adopting appropriate forward declaration and minimize inefficiency as much as possible.

Secondly, it is important to have good division of work and cooperation. At the beginning, we do not have enough communication with one another, and we wasted much times on reading and understanding other's code. We should try to communicate more. Also, we found it's actually easy to make the division of work based on MVC pattern (each is responsible for one branch), so that we can avoid to waste too much time to read one another's code. (as Model, View, Controller do not have much interaction at all)

2. What would you have done dierently if you had the chance to start over?

We would spend more time on hint method in class Board, as it is actually much more complicated than we expected, and we would like to add bridge pattern to our design. We believe it is truly a good design and a great habit to minimize the recompilation, as it definitely will help us during the future coding when we are employed. Although the reduction compilation currently seems negligible, it will matter a lot when we are deal with gigantic amount of coding.