QUADRIS DESIGN DOCUMENT

CS246 Final Project

Jiamian Shi, Yongyuan Bai, Wenlin Xu

Dec 2017

1. INTRODUCTION
2. OVERVIEW
3. **Changes made to initial plan**When creating the planning document, we believe the game is very similar to Reversi, except that we have to generate blocks and drop them down on the grid. However, as we implement the grid class we find it is more organized to add a quadris class to keep track of all blocks separately. As a result, the grid class is only responsible for cells on the broad and it works together with class quadris to accomplish the functionalities we initially designed for class grid.

In addition, since all blocks share the same movement functions, we find it easier to implement methods with multiple cases for different block types, instead of making subclass for each block type.

Similarly, we wrap all necessary level methods in the level class so the level subclasses are no longer needed.

1. **Description of overall structure**Overall, we applied the observer pattern in our structure. Cells are both the Subjects   
   Same as the subject class in the observer pattern, our subjects class maintains a list of observers that can subscribe to the subject. Changes in subjects will notify observers.

Observers

The observer class coupled together with subject class to form the basis of observer pattern.

Quadris

The Quadris class wraps up the functions we need to execute the commands. It maintains a list of blocks since a new game begins, so it can keep track of the state of each block and update score when one block is cleared. To achieve extra credit features, other fields such as grid, level, score, graphicsdisplay and textdisplay are made as unique pointers.

Grid

The Grid class represents the grid where the Quadris game is played on. It uses a 2D array of cells. Once a block is dropped, it will go through all cells on grid, set the cells where the new block is dropped to a specific shape, and clear any line which is full.

Block

The Block class represents seven different block types. It maintains a list of Info, which contains the positions of active cells and the shape for this block. Once a movement is made, the list Info get updated, and the block knows its new position of each active cell on grid.

Cell

The Cell class represents each Cell on the grid. It stores the position of the cell and the type of block it belongs to. It is a concrete subject class which will notify TextDisplay and GraphicsDisplay when it changes state.

Display

The Display class is a base class for the different ways the game is shown to the user, via textdisplay and graphicsdisplay.

TextDisplay

The TextDisplay class is responsible for the standard text output of the program. It is an observer and will print the state of the grid after executing each command.

GraphicsDisplay

The GraphicsDisplay class creates the Xwindow of the program. It is an observer and will update the Xwindow after each command.

Level

The Level class represents the different difficulty levels available to the game. It is responsible for generating a next block for the game. There is a sequence stored in the class for generating blocks, and all different levels have a unique score generating function.

Score

The Score class stores the current score and high score of the running game.

Interpreter

The Interpreter class parses commands as strings, and then passes it to the class Quadris to call the relative methods. It uses a Trienode in autocomplete class which will complete the command with a valid prefix.

1. UPDATED UML
2. DESIGN

1. RESILIENCE TO CHANGE  
   Our program will be able to support various changes to the program specification.

To introduce new block types, we can simply add a new shape to the class shape, and add the constructor of this new block type to the class block because all different blocks share the same movement functions, and no additional methods or classes need to be added to the class block. Then we should add cases to methods in class level to adapt to this new type of block.  
To change the board size, we can modify the WIDTH, HEIGHT and BUFFER\_HEIGHT in constants.

Other possible changes are specified in [answer to questions].

1. ANSWERS TO QUESTIONS
2. **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 the grid class, we stored a list of all blocks on the grid in the order of their drops. To allow for some generated blocks to disappear from the screen if not cleared before 10 more blocks have fallen, a field count of type integer can be added to the class Block. As the game begins, the first block has its count set to 0, and every time we drop a new block, the filed count will be incremented by 1. In this way, the field count can also represent the index of each block in the block list stored in grid. Suppose the count of current block is n, after this block has fallen, we check the block with index (n-10) in the block list, and clear it if it is still on the grid.

It can be easily confined to more advanced levels because the only differences of levels are the generating patterns of new blocks and the scoring system. Both the field “count” and the list in grid will not be affected in this condition, so we have no worries in advanced levels.

1. **How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?**In the initial plan we decided to have level subclasses so that we can simply add a new level subclass to accommodate the possibility of introducing additional levels. However, as we implement the level class we find it not necessary to have level subclasses. Instead we use if-else conditions to implement methods for different levels. In this case, to introduce additional levels we need to add more conditions to methods implemented in the level class. Also, we still need to modify the levelup and leveldown functions in quadric class to recognize the new levels.
2. **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.**

As all commands are stored in TrieNodes, we can simply insert this new command into TrieNode, and modify the interpreter to handle it. Also, the Blocks need to be modified to add methods to accommodate the new command.  
It is not difficult to sort through commands, add commands, and remove commands through TrieNodes. To change existing command names, we should modify the command handling function to change the map of string and command by deleting the string the command previously mapped to, and map it to the new one.

1. EXTRA CREDIT FETURES

For extra credits, we apply smart pointers in the class quadris. With these smart pointers, the allocated heap memory will be automatically deleted so we do not need to worry about memory leak.

1. FINAL QUESTIONS
2. **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?**

First, through this project we learned to cooperate with others to achieve a common goal. As individuals we all have different thoughts about our project, and to work as a team we should learn to listen to others’ opinions and adapt the most organized and efficient method. Also, to achieve efficiency, we discuss about each other’s strengths and weaknesses to make sure everyone is working on what they are good at.

In addition, it is the first time we use a shared private repository on github and we all find it really helpful. It is easy to contribute to our project separately and get track of how many code changes is made every day.

1. **What would you have done differently if you had the chance to start over?**

If we have time to start over this project, first we will start earlier because this time we experienced significant time constraints. We failed to follow the deadlines in the initial plan, so in the end we did not have enough time to complete all the features of this game. For example, we had considered the method for hint in the beginning, but as DD2 approaches we suffered a lot from debugging the basic commands and finally give it up.

In addition, for the class interpreter, we should implement a Trienode class to search for prefixes instead of hardcoding. The reason that it is better than our current implementation is that adding, removing and editing commands will be much more flexible for the programmer, and it is easier to handle a larger number of commands without too many changes in the original implementation.