CS 246 Final Project

Biquadris

Yu Li: y872li (20603929)
Ze Si Dong Li: zsdli (20583608)
Yitian Wu: y468wu(20631864)

Design

Observer Pattern:
Subject: Cell
Observer: Graphic Display & Text Display

Factory Pattern:
Used on the levels for generating blocks

Project Schedule

Date	Activities	Person
November 17-18	Building classes for the project Core class - Cell and Grid Class - levelscore	Yu Li/ Ze Si Dong Li/ Yitian Wu
November 19	Core Class - Block with basic methods (excluding rotation) Core Class - Quadris with basic command (only one command for left, right, drop, and restart)	Yitian Wu/ Ze Si Dong Li
November 20-21	Core Class - Level 0, 1, 2	Yu Li/ Yitian Wu
November 22	Core Class - Player without scoring	Ze Si Dong Li/ Yu Li
November 23	Core Class - Subject, Observer, TextDisplay	Yu Li/ Ze Si Dong Li/ Yitian Wu
November 24	Core Classes Testing	Yitian Wu
November 25	Class - Level 3, 4	Yu Li/ Ze Si Dong Li
November 26	Graphical Display Class - Subject, Observer, GraphicDisplay	Yu Li/ Yitian Wu
November 27	Class - Player with correct scoring algorithm	Yitian Wu/ Ze Si Dong Li
November 28	Class - Block (adding function of rotation)	Yu Li/ Yitian Wu/ Ze Si Dong Li

November 29	Core Class - Quadris supporting all command inputs including command input variants	Yu Li/ Yitian Wu/ Ze Si Dong Li
November 30	Main with additional command-line arguments	Yu Li/ Yitian Wu/ Ze Si Dong Li
December 1	Testing and debugging	Yu Li/ Yitian Wu/ Ze Si Dong Li
December 2	Testing and debugging	Yu Li/ Yitian Wu/ Ze Si Dong Li

Anticipated Plan for the Questions

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?

Answer: our plan is to add an integer time field and an isDisappear boolean field to each block. Grid will keep track of the special blocks when they drop. When a new block drops, the grid will notify each special block to update their time field. When any of the block's time reaches 10, the grid will remove the block and update each cell.

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

Answer: We are creating levels with the factory method design pattern. When adding additional levels, we can just create more concrete level classes that implement the Level interface. Each level will generate blocks based on its logic. If one is not introducing features that did not exist in the game before, one would only need to recompile the level class. Using more forward declarations instead of #inclue can also reduce the number of build times.

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?

Answer: One can utilize the visitor pattern. The visitor will let us define a new effect without changing the other classes of the elements which the effect operates on. The visitor effect class will have a pure virtual visit() method. Each concrete effect subclass will have a visit() method that accepts an argument, where it could be a pointer or reference to the original element derived class. In this case, it could be a pointer to the level derived classes. Each concrete derived class will then have an accept() method where the argument is a pointer or reference to a visitor. This pattern makes adding new effects easy, just simply add a new visitor class, let the visitor class visit the level classes, and the level classes will accept the visitor.

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, th adding a new command probably means adding a new featurate, 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.

Answer: We can have a map to contain all the commands. The commands with the same functionality but different names can have different keys but same value. When changing the command names, old names will be removed from the map and new names will be put into the map with the same old values. Since we can just modify the key instead of the value, it would be fairly easy for our system to support a renamed command while calling the command based on the value in the map. To support macros, the values in the map can be a list of names. When a macro is added, the key is the new macro name, the the value is the list of functionalities one would run.