CS246 – Assignment 5 (Quadris)

Yuan Gao – 20429092 – y65gao
Planning Document

Abstract

This document summarizes the plan of implementation for the game of Quadris, and breaks down the process. The sole contributor to this project is Yuan Gao, with no other team member. Please note that all deadlines in this document are tentative. The primary tasks in the project are the following: construct the UML diagram of the game, creating this document, writing the actual source code, and completing another design document that outlines the final design of the project and compares the differences (if any) of the final result to the plan submitted with this document. The final deadline of the project is April 4th, 11:59 PM.

Table of Tasks and Deadlines

The following table describes the tentative schedule for the completion of each anticipated step of the project.

Task	Description	Deadline
Initial UML Document	The first version of the UML document attempts to model which classes/methods would potentially be needed to implement the full program. This UML is expected to not be completely accurate and may have large rooms for error/modification during the implementation process.	3/23/2014
Planning Document	The planning document is this document. It describes the plan of attack and schedules potential tasks to be completed. It also acts as a to-do list to make sure every component is completed.	3/23/2014
Creation of the Board, Blocks, and Cells – Header files and implementation	The implementation of Quadris will attempt to follow very closely to a similar question in Assignment 4 that also involved Xwindow: the game of Light's Out, which had a board and cells. Now there are extra block objects that occupy the cells instead of just a single object per cell.	3/26/2014
Creation of Score, Game, and Interpreter	After the creation of the board, blocks, and cells are complete, the next step would be to create the actual game, and implement an interpreter that can communicate between all the blocks and the board. Score will be a separate class that tracks the player's score. Game logic will be implemented as well.	3/30/2014
Creation of the TextDisplay, and GraphicDisplay	Similarly to the game of Light's Out a Text Display and Graphics Display will both be implemented. The Text Display uses the terminal to display text that conveys the game. The Graphics Display uses Xwindow.	4/1/2014

Creation of the Command-line Interface	Similar to the driver files written in previous assignments, this is going to be basically main.cc – where user input will first be evaluated when running the program to setup the game.	4/1/2014
Bonus Features	Bonus features will be attempted to be added. As this is a solo project, it is uncertain whether many bonus features will be included. The only currently planned one is the ability to skip a block if the user has enough tokens. Tokens are generated by completing a successful 4-line clear.	4/2/2014
Comments and Documentation	Review all the code and write comments wherever the code is obscure or a particular design pattern is of interest.	4/3/2014
Final Design Document	The final design document covers all aspects of the implementation of the program at a high level.	4/3/2014
Memory Leak, Testing, Final submission	Although testing is expected to be done throughout the design process, the project is expected to be submitted on the final due date, after extensive testing for memory leaks, bugs, or other weird behavior.	4/4/2014

Project Specification Questions

How could you design your system to make sure that only these seven kinds of blocks can be generated, and in particular, that non-standard configurations (where, for example, the four pieces are not adjacent) cannot be produced?

We start with a single abstract Block class with common virtual methods that can then be extended to "subblock", specifically, the 7 types of blocks that are possible, each in their own class. Since the game will use Blocks, only these 7 types of blocks are possible to use. This also means that non-standard configurations are not possible because each sub-block class is a specific, predefined design. The game will never be able to deviate from this design unless some code was buggy.

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?

There could be a global game counter that counts the number of blocks that has fallen, and each block is assigned a number as soon as it's generated. The special blocks that disappear will have an additional Boolean that indicates if it can disappear or not. Every time a new block is created, the special blocks on the grid check if the number of the block that just got generated and the difference between them is larger than 10. If it is, then the older special block gets removed from the grid.

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

It can be easily confined to more advanced levels. The method for the block being deleted will only be called if the Boolean of level being larger than [x] for example is true.

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

If difficulty levels scale only with the relative probability of certain blocks being harder to get, it is possible to make a separate class where the sole purpose of it is to generate the next block. To implement a new level, main.cc needs to recognize the command, and so the only things that need to be recompiled is the block generator class, and the main program.

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.)

To add a new command, because the Command interpreter handles this part, and then the Abstract Block then does whatever the command needs to do, three files, including the command line file, would need to be modified. Another way to design the system would be to have each command be in their own separate class and be subclasses of the Interpreter, however this requires a new command class to be created each time which is not very efficient.

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)?

To change existing command names, because main.cc maps input to commands, it would have to modified to accommodate and either map the command to the additional command, or it would need to delete the previous string the command used to be mapped to and map it to a new one, or it could modify the string that the command was mapped to. This is not too difficult to implement.