Assignment 5

Quadris

CS246

Fall 2017

Roshan Rishav Huiyi Cai Jiawei Yu

Plan of Action:

The following are the broad tasks that we have divided the project into:

- Brainstorm and prepare the initial UML.
- Prepare the plan of action document.
- Implement the code, following the initial UML.
- Code check and testing.

Schedule:

Sl No.:	Task:	To be done by:	Due:
1.	UML design decision and brainstorming.	Charlie Yu, Judy Cai, Roshan Rishav	22nd Nov, 2017
2.	Final UML implementation.	Charlie Yu	25th Nov, 2017
3.	Initializing the git repository and implementing the base header files.	Judy Cai	25th Nov, 2017
`4.	Documentation and plan of action file for due date 1.	Roshan Rishav	25th Nov, 2017
5.	Final review for due date 1.	Charlie Yu, Judy Cai, Roshan Rishav	26th Nov, 2017
6.	Implement the base classes, with all of the basic methods.	Judy Cai	27rd Nov, 2017
7.	Implement the blocks, with all of the required methods.	Charlie Yu	28rd Nov, 2017
8.	Implement the board, to support at least level 0.	Roshan Rishav	28rd Nov, 2017
9.	Implement the GraphicsDisplay, to support at least level 0.	Judy Cai	28rd Nov, 2017
10.	Implement the main.cc to support the command interpreter.	Charlie Yu	29rd Nov, 2017

11.	Test level 0.	Roshan Rishav	29rd Nov, 2017
12.	Add functionality for levels 1, 2	Charlie Yu, Judy Cai, Roshan Rishav	30rd Nov, 2017
13.	Test the functionalities implemented so far.	Roshan Rishav	30rd Nov, 2017
14.	Implement levels 3, 4.	Charlie Yu, Judy Cai, Roshan Rishav	1rd Nov, 2017
15.	Test the functionalities implemented so far.	Judy Cai	2rd Nov, 2017
16	Implement additional features.	Charlie Yu, Judy Cai	2rd Nov, 2017
17.	Final Testing.	Roshan Rishav	3rd Nov, 2017

Questions:

Q1. 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?

To allow for select blocks to disappear from the screen if not cleared before 10 more blocks have fallen, we can set a class field such as “bool destructible” in the cell class to indicate whether it is a cell that will eventually disappear. Then, when a block that is meant to disappear is created, it will initialize its vector of cells to cells that set destructible as true and another integer field such as “int timer” with an initial value of 10.

Hence, whenever a block is dropped, the main board will iterate through its grid of cell pointers, check that it is not empty and is a destructible cell, and decrease the value of the timer of the cells by one each time through a cell class method.

Following this, there will be a check to see whether any destructible cells have timer equal to 0, and will be deleted accordingly.

This can also be confined to more advanced levels by simply altering the implementation of the generateBlock() method of the advanced levels to generate blocks with destructible cells.

Q2. How could you design your program to accommodate the possibility of introducing additional levels into the system, with minimum recompilation?

To accommodate the possibility of introducing additional levels into the system, we can have a separate class for “Level”, and then using it as the base class, we will implement separate child classes for each of the levels. So whenever we want to implement a new level, all we will have to do will be add another level subclass with all of the required specifications of the levels in it. Apart from this, we will also use good design patterns when including/forwards referencing the header files/classes, to further reduce the required recompilation. Using the pimpl idiom, we can minimize compilation dependencies when private fields are added to the base level class to accommodate for extra features.

Q3. 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 accommodate the addition of new command names, or changes to existing command names, we can have a separate header files with all of the “strings”/ “command names”, and will use these strings as the names of the respective commands. By doing this, whenever a user wants to change the names of the commands, we can simply modify the string header file accordingly.

By using the above mentioned design, it will not be very hard to implement a command with which a user can rename the exiting commands.

To support a “macro” language, we can implement something like a command “queue” in a separate class, and can have a separate command to initialize it. So whenever, the user enters this command, the i/o will ask the user to input a sequence of commands, which will be stored in the queue, and once the user is done, the i/o will then ask for a name for the queue, and will store the name of the queue in a very similar fashion to that of any other command. Now, whenever the user will enter the new macro command, all of the commands in the queue will be executed in proper order (only if the command can be executed legally).