6: A Variation of Board

CSCI 4526 / 6626 Fall 2022

1 Goals

- To use derivation to create a variation of the Sudoku game board.
- To show me that you can use UML class diagrams.
- To experience the interaction between the constructors of the base and derived classes.

2 Variation: A Board with two more constraints.

Several variations of Sudoku have been invented. Some are easy to implement, others quite difficult. In this assignment you will implement an easy variation. A different variant will come next week.

Diagonal Sudoku. In a diagonal game, almost everything is the same as in a traditional Sudoku board: the Squares, Clusters, and the user interface. The DiagBoard class has one addition: two more constraints, and therefore, two more clusters, going diagonally across the board from one upper corner to the0 opposite lower corner. The only legal size for a DiagBoard is 9.

The format of the input file will be the same, but now the puzzle type on the first line can be either a 't' (for traditional) or a 'd' (for diagonal). The next 9 lines of the file will contain the puzzle, as before.

- Read the first line of the file as usual. If the type code is 't,' dynamically allocate a traditional Board and attach it to the Board*. For a 'd', allocate a DiagBoard. You should not need any other changes in Game.
- Change your cluster-type enumeration and its parallel array to include DIAG for "diagonal".
- Change your Board constructor to take a parameter, which will be the number of clusters to allocate. The default is 27, for traditional boards, as before. The other possible value is 29.
- Derive a DiagBoard class from Board. In the DiagBoard constructor, initialize the base class, passing N and the open stream to the Board constructor in a ctor. In the body of the DiagBoard constructor, create the two diagonal clusters and add them to the Board's array.

  Note that the result will be to add one more cluster to each Square on the diagonals, and two more clusters to the center Square. This should all work with no fuss because the Squares use vector to store the Cluster*s and a Square shoops all of its Clusters each time a mark is made. This is where the combination of OO and careful design pays off.

3 Final Project Requirements

I want you to use all the techniques below. In some cases, you must choose where and how to use them. You should begin now to use them in your code.

1. Use a static class variable.
2. Use ctors in the required places and also in some of the *optional* places.
3. Use default parameters somewhere.
4. Use const wherever you can in your program. (Const parameters, const this, const local variables, const return type, const global information.)
5. Use an enum class

4 Due October 29

Implementing the new part of the project should be a very quick and easy job. Submit these things, in the order given, electronically if possible, and in the form of a single zip file.

The game. Use the input file puzd.txt to initialize your board and submit output from Board::print() just after construction. Verify that your code creates correct diagonal clusters and shoops them appropriately.

Make several moves involving diagonal squares, including the center, and submit output again.

Retest your Program 6 to make sure it still works with traditional sudoku puzzles.

Hand in a simplified UML diagram showing just the classes and their relationships (boxes, names, and links, no other details). Focus on getting the relationships right.