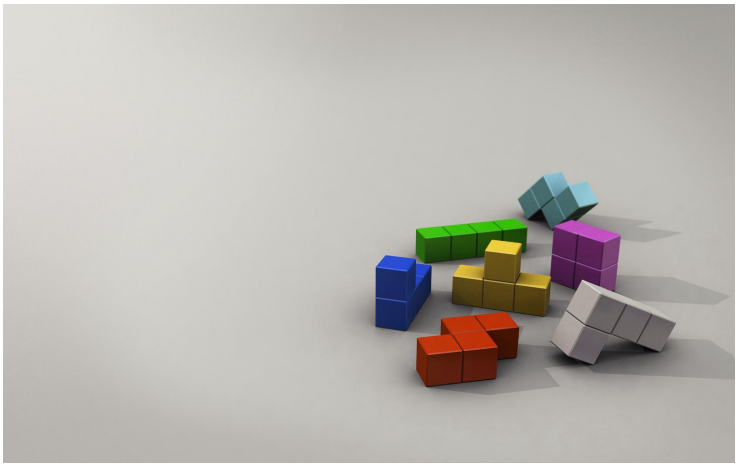
CS 246
FINAL PROJECT



by:
xyuan
x445wang

source: http://stylefavor.com/wp-content/uploads/2012/10/Tetris-Wallpaper-3D.jpg

Plan of Attack

All Classes we need:
Cell, Board, Display, Score, NextBlock(and NextBlockImpl), Block(and all its subclasses) CommandInterpretor

Estimated Completion Date: Next Wednesday (2nd, Apr)

What to do in order:

1. Cell
2. Board
3. Display
4. Score
5. NextBlock(and NextBlockImpl)
6. Block(and all its subclasses)
7. CommandInterpretor

Reasons of this order:

1. The implementation of cells should come first simply because it would lay the foundation for the whole program. Because eventually, we would need to store most of the information in the cell, e.g. how many active neighbours are left and what kind of block is this cell from. Thus we need the information from the cell to perform further instructions.

2. Board contains a two-dimensional array of cells that make up the grid. After we complete this class, we will be able to achieve the simplest task, that is, setting the board to default status(i.e. what it is at the beginning of the game).

3. Class Display will let us show how the board(grid) actually looks like in real time and would help us in debugging in the future.

4. Score and NextBlock are relatively easy to implement and relatively independent from the main program. Score class keeps track of the highest score and the current score of the game and would notify the Display in real time. And NextBlock is responsible for generating a new block depending on the difficulty level of the game; it is only required when we need to manipulate the block class. therefore, we will need to complete this before we move on to the most tedious part, block.

5. The main reason we wait until this step to implement the Block class is because Block requires many interactions with other classes such as Cell and Board. Thus, we are only implementing Block now because Cell and Board are ready to be used. Also, the Display class should be working functionally so that we can better visualize the movement of

blocks, especially rotation of different shapes of blocks simply because they all have unique behaviours, in the game and identify any incorrect implementation.

6. We leave the CommandInterpretor to the very end because the function of the this class is very independent from the rest of the program. The purpose of this class is to incorporate different ways clients are able to interact with the game, such as autocompletion of the command and personalize the command. However, the implementation of the class would still give us the standard command, which means we don't need to worry about the exact implementation of this class that means we can and we should leave this to the very end.

***Each partner's task*:**

Xueling Yuan: Board, Cell & NextBlock
Xiangkang Wang: Display, Score & CommandInterpretor
Together: Block and its subclasses

***Question*:**

- **Question: 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?**

- Answer: Use Inheritance, that means we have an abstract superclass called Block, which has 7 subclasses (Iblock, Jblock, Lblock, Oblock, Sblock, Zblock,Tblock),. Therefore, only the abovementioned blocks(shapes) would be created and no other blocks (non-standard configurations) will be created. In addition, each block has an array of 4 cell pointers to store the corresponding cell in the grid.

- **Question: 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: We need to introduce two more variables into the implementation. First, we keep track of the when each new cell that is entered the grid, so that we know which cells to be deleted when another block is entered in the grid. Then, we use a variable called to MaxDelete to keep track of how many blocks are allowed to remain in the grid. In this case, MaxDelete would be 10, because each block should be cleared before 10 more blocks are generated. And we are going to set the default value as INT_MAX. This implementation would work for every difficulty level.

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

- Answer: We can achieve this by utilizing the bridge design pattern. when we are talking about changing the possibility of types of blocks that are being generated. Thus, in the NextBlock class, we are going to use the bridge design pattern. We introduce the NextBlockImpt class which incorporates the actual implementation of different levels and NextBlock contains a pointer to the NextBlockImpt to access to the implementation of different levels. Eventually, we only need to recompile NextBlockImpt and achieve the minimum recompilation especially for the client.

- **Question: 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)?**

- Answer: In other words, we need to recognize different command efficiently. To achieve that, we can adopt the technique that was used in a2q4 where we construct a tree to store words. In this case, we can add all the words in the tree and add the type of command at the end of the word so that we are able to recognize the command type as long as we can differentiate them. This would allow us to make minimal changes to the source code because no more new lines of code needs to be added if we try to add new command to the game or modify the existing command, all we have to do is just to add the command to the tree and we will be able to recognize them.