|  |  |
| --- | --- |
| **Project 3** | |
|  | Have we designated a daily secretary? |
|  | Have we decided on a division of project tasks?   * Secretary * Tester * Programming Pair |
|  | *Have we decided on data structures for each module?* |
|  | Read through specs concurrently while working on sections. |
|  | Have we downloaded the most recent version of project from Git? |
|  | Have we written javadoc comments for our implementations today? |
|  | Have we written tests for all public methods implemented today? |
|  | Have we implemented equals(), toString(), hashCode() for every class? |
|  | Have we kept all our exceptions’ messages in a separate central file? |
|  | Have we pushed the changes from today into Git? |

***Day 1-2***

*8/02/13 Clancy meeting*

* Tree is not explicit, more implicit because we are considering different move choices
* Javadoc good idea as part of write-up
* Put discussions w/ Clancy in write-up
* GitHub used by Clancy associates
* Suggestions: Write so that you can plug & unplug different algorithms and data structures
* Take into account that Blocks could consist of more than 4 grid points
* In regards to storing exceptions’ messages, think about using isOK
* Tray representation: Should the tray be stored as a list of blocks/empty spaces to optimize move generation, or should the locations in the tray be represented explicitly? If the former, should blocks/spaces in the list be sorted?
  + Prior to each move, the program must check whether the desired configuration has been achieved. What tray representation optimizes this operation? If this representation is incompatible with implementations that optimize move generation, how should the conflict be resolved?
  + You are to include an isOK method with your tray class. When the tray class's debugging option is enabled, a call to isOK should accompany each change to objects in the class. The isOK method should throw IllegalStateException with an informative message if it finds a problem.
  + Each block consists of 1, 2 or 4 points (Point Class)
  + Define a block as two end points in the board. (Block Class)
  + Define the tray as arraylist of blocks (Tray Class)
  + Referencing each block in the tray needs to be efficient (not by a list, maybe hashing)
* Once it has a collection of possible next moves, the program will choose one to examine next. Should the tree of possible move sequences be processed depth first, breadth first, or some other way?
* Should block moves of more than one space be considered? Why or why not?
  + With our current configuration, moving blocks would not very difficult.
* The program needs to make and unmake moves. Again, a representation that optimizes these operations may not be so good for others. Determine how to evaluate tradeoffs among representations.
* The program must detect configurations that have previously been seen in order to avoid infinite cycling. Hashing is a good technique to apply here. What's a good hash function for configurations? The default limits for Java memory allocation may limit the maximum number of configurations that the table can contain. How can this constraint be accommodated, and what effect does it have on other operations?
  + An experiment we recommend is to determine how many configurations you can add to a hash table before you run out of memory.
  + Have to use HashSet in order to detect configurations that have previously been seen
  + When we recursively search for a conclusion, we have to memoize. The entire tray as a whole will be hashed.

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

***Day 3***

*8/4/13 Group Meeting*

* Download easy and hard puzzles from online
* GitHub registration / push access
  + Installation and user configuration
  + Project Import (downloading.. local copy / diff from dropbox)
  + Adding files
  + staging=saving / committing=local (hard to delete) / pushing=remote
  + synchronizing / pulling (checks conflicts)
  + comparison revert / hard reset
  + Branching (don't push) / merging (delete afterwards) / resolving conflicts

*Design & Coding*

* Point class
  + Point class creates point from two integers
  + Immutable: rowIdx (row index) and colIdx (column index) are final values
  + Constructor
    - Check validity of arguments
* isLowerRightTo method & isUpperLeftTo method
* Board class
  + Block class constructs a block from four integers, by computing its width and height from these inputs (this allows us to make moving blocks more efficient).
  + Constructor
    - Check validity of arguments
* Tray Class
  + Constructor
    - Check validity of arguments
  + rowSize and colSize are immutable
  + addBlock method
    - Make private because no more blocks added after initialization
    - Check if given block can be legally added to the tray
    - checkBlockAddition method: If given arraylist of blocks in constructor, check for validity and then use as tray’s arraylist
  + move method
    - Moves block in tray at blockIdx (block index in tray’s arraylist) to new upper-left coordinate.
    - *Design*: The block’s upper-left point is changed to the new one and then its addition is checked for validity by checkBlockAddition
* toString() method for Point, Board and Tray class
* QOTD: When are two trays equal??
  + Trays are same size and contain same number of blocks

Tray is equal when trays have blocks of same size in same positions (picture of trays are the same)

* **DESIGN:** Using arraylist to store blocks would have been problematic if not for the fact that an arraylist can be converted into a set by placing the list as the argument when constructing a set.
* **Testing:** Write tests in BoardTest, PointTest, TrayTest

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

***Day 4***

* Solver Class
  + Main class
    - Convert given initial Tray file into Tray
    - Convert final configuration file into collection of blocks
  + Solve() method
    - trySolving method
    - Initially DFS
* CopyOnWriteArraySet change
  + Int variables changed to short to conserve space
  + Removed isOccupied
* Method to improve how Solver makes moves (Alternate to current DFS method)

1. Get corresponding blocks
2. Try move them first before my other shapes
3. For each possible move, evaluate the new configuration’s measure to the good
4. Do the move with the lowest measure first