Group 66
Tyler Allen
Brenden Arias
Homework 8 Report

This program utilizes 5 java files: Solver.java, Tray.java, Block.java, Move.java, and Coordinates.java.

Solver.java contains the main logic behind file reading, Tray object construction, and solution finding. It starts by taking in a series of command line arguments. It filters out any options and executes those, and stores any non options as file names. If there are not enough file names (2), the program will exit. Solver then reads in the data from file one into a list, which is then constructed into a Tray object containing the initial tray configuration. Solver reads in the data from the second file, and that is constructed into a second tray, the same dimensions of the first, containing the solution goal data. Solver then calls the solve method.

The solve method, inside Solver.java takes in a tray containing the initial configuration and a tray containing the goal configuration. Solver then performs the following steps:
Initializes a list of moves (See Move.java) with the moves available in the initial tray configuration.

In a loop:

Copies the initial tray into a temp tray.
Applies the first move in the list of available moves
Checks if this tray configuration is one previously tested
(If it is, skip it)
If this configuration contains the goal
Return the moves needed to reach this tray
Otherwise, add this tray to a list of previously completed trays.
And add all the possible moves from this tray onto the list of available moves

If the solver is unable to find a possible solution (Runs out of moves), it will return null.

Tray.java is an object representation of a tray of blocks. It contains dimensions, a list of blocks that it contains, and a list of moves that have been performed on it. It also has all the methods necessary to copy trays, build itself from a list of strings, return a list of possible moves, have moves performed on it, check if it contains a given block or list of blocks, and is able to ensure that it contains a valid configuration of blocks, and that blocks are unable to be moved in invalid ways.

Block.java is an object representation of a block. It contains a height, width, and coordinate position (Coordinates.java). It also contains the methods to: construct itself from 4 integers, copy a given block, move it's location given a coordinate or x and y, and can check if it overlaps with a given block.

Coordinates.java is simply an object to store an x, y position. It can create itself given an x, y, or it can copy a given coordinate object.

Move.java is an object for storing movements performed on blocks. It stores a block and a coordinate representing a movement for that block. It can also contain a move representing a previous move necessary for reaching this move.

You can see the JavaDoc, which can be found in “Homework 8/dist/javadoc/index.html” for additional and more specific information on each method.

In terms of efficiency, the current solver is not incredibly efficient, but we are currently looking into ways to improve it. A specific alternative we are looking into is the use of hashing for storing the previous attempted configurations in the solve method, this could potentially speed up runtime significantly.

The program typically uses between 1 GB and 2 GB of ram on average problems, though could use significantly more or less depending on the problem. Blockado for example, uses approximately 1.3 GB of ram during execution.

The program’s runtime can vary drasticly depending on the difficulty of the problem. For easy problems the runtime can be as low as a second. For medium problems the runtime is normally 10-30 seconds. For extremely difficult problems, the runtime can be as high as 30 mins.

One big thing we’d like to improve is the runtime for extremely large problems. The main source of this issues comes from checking if the tray has been previously tested before. Currently, the program must check all previous trays.equals(new tray), which is extremely inefficient. We’d like to implement a different solution possibly involving hashing, in the future.

To enable debug output, pass any desired arguments into the program at runtime. The possible arguments are: -h: Lists possible options, -oruntime: Prints the runtime of the program, -omaxmemory: Prints the max memory usage, -ototaltrayschecked: outputs the number of trays attempted before a solution was found, -onumberofmoves: prints the number of moves required to reach the solution.