Principles of Object Oriented Programming – Assignment 3
Bonus Method Description

מגישים:
איתי בוגנים, 305278384
סהר ויה, 205583453

Description:

The Method shuffles the board to a solvable permutation of the bricks by mimicking a movement (As A user would move) of bricks across the board in the following manner:

1. Calls the shuffle board method with a number of wanted brick movement to make, In order to make sure the board is appropriately shuffled, the number of wanted movement as we defined is: 10,000 times the squared board size.
2. Randomizes a neighboring brick to the current empty brick on board by randomizing a movement offset from the empty brick. (-boardSize – UP, +boardSize-DOWN, -1 -LEFT, +1-RIGHT).
3. Checks (By using the helper method isLegalBoardPlace) that the neighboring brick that has been randomized is in a legal board place (within the board size limitations, 0 to squared board size).
4. Checks (By using the helper method canMove) that the neighboring brick that has been randomized can be legally moved to the empty bricks place.
5. If the checks have been failed, randomizes another neighboring brick by iteration until a legal neighboring brick has been found.
6. If both checks passed, swaps the empty bricks place with the neighboring brick place.
7. Updates the empty bricks current index and decreases the shuffle moves remaining to make.
8. Iterates the entire process until all wanted shuffle moves have been made.
9. The board is now shuffled and solvable always. (If the end-user tracks back the shuffle movements made it will solve the board).

Note: No moves are made by "Picking" a brick and replacing it with another, all moves are made by sliding the bricks to an open spot (in a legal In-Game move).

Method call – Calls the method with 10,000 times square board size wanted brick moves.

```
if(shuffleMethod =="Auto shuffle") //Dynamic shuffle
    shuffleBoard(10000*squareBoardSize);
else systemShuffle(); //Shuffle from list of boards loaded from boards.csv
```

Method operation – Randomizes a neighboring brick to the empty brick and swapping with the empty spot If both checks for a legal board position and a legal board move has passed.

```
/*
 * A dynamic shuffle method, locates a neighboring brick to the empty index and randomizes a move over the board by iteration.
 * @param shuffleMoves - the number of moves over the board that should be made.
 */
private void shuffleBoard(int shuffleMoves)
{
    Random rand = new Random();
    int[] neighborOffsets = { -boardSize, +boardSize, -1, +1 }; //Possible moves set - up down left right
    while (shuffleMoves > 0) {
        int neighborBrick;
        do
        {
            neighborBrick = emptyBrickIndex + neighborOffsets[rand.nextInt(4)];
        }
        while (!isLegalBoardPlace(neighborBrick) || !canMove(neighborBrick, emptyBrickIndex));
        Collections.swap(PuzzlePanel.bricks, neighborBrick, emptyBrickIndex);
        emptyBrickIndex= neighborBrick;
        shuffleMoves--;
    }
}
```

Legal Randomized Brick spot checks – checks that the neighboring brick randomizes is within board size limits and that that it can be moved to the empty spot

```
/*
 * Check if a brick can be moves determined by the brick clicked and the empty brick locations.
 * @param - clickedIndex - The brick wished to be moved, the brick that was allegedly clicked.
 * @param emptyIndex - The location of the empty brick, the place to check if the clicked brick can be moved to.
 * @return True if brick can be moves false otherwise.
 */
private boolean canMove(int clickedIndex, int emptyIndex)
{
    return ((clickedIndex - 1 == emptyIndex && clickedIndex%boardSize != 0) || (clickedIndex + 1 == emptyIndex && (clickedIndex+1)%boardSize != 0)
            || (clickedIndex - (boardSize) == emptyIndex) || (clickedIndex + (boardSize)  == emptyIndex));
}
```

```
/*
 * Checks that a place is within the limits of the board.
 *@param place - that place to check if is part of the board.
 *@return True if the place is a legal place in the board False otherwise.
 */
private boolean isLegalBoardPlace(int place)
{
    return ((place>=0) && (place<squareBoardSize));
}
```