* **Jeffrey Kennedy – Project Lead**
  + As the project lead, I was responsible for organizing meetings, delegating project tasks, and managing conflict.
  + As well, I took responsibility for creating a package of java code that will randomly generate Sudoku puzzles of varying difficulty.
    - This means the Board.java, House.java, and Tile.java files.
  + I constructed the three objects with the intention of interfacing them directly with the GUI – meaning each tile displayed on the board would be tied to a specific tile in the Board and the Board would be the object being interacted with by the rest of the app. However, once work on the GUI got underway, it became clear that it would be easier for the GUI designers to develop something of their own to interact with. Thus, I repurposed the files into simply being a random puzzle generator.
  + I researched algorithms for puzzle generation and discovered that the most common procedure is to remove tiles from a puzzle which is already completed (and valid) in an intelligent fashion that determines difficulty.
    - This is implemented in Board.java. See the comments for details.
  + The easiest way to generate a large variety of puzzles turned out to be using one hard-coded puzzle and then shuffling it up to a thousand times using several different types exchange (already validated by other researchers)
    - This is implemented in Board.java. Specifically, in the “shuffle()” method.
  + This underlying structure of Board -> House -> Tile provides an opportunity for future improvements to the game, should we wish to continue its maintenance.
* **Charles Lowell – Processing Designer**
  + I researched Sudoku solving algorithms and decided to reduce Sudoku to exact cover and use Knuth's Dancing Links algorithm to find solutions.
    - The latter is in class DLX.
    - Dancing Links is an almost completely stupid backtracking brute-force search made fast by its use of a clever toroidal quadruply-linked list, which I implemented in the DLXMatrix and DLXNode classes.
  + Next I wrote code to perform the reduction and interface with DLX
    - This is in class SudokuSolver.
  + I also wrote a small program to test the solver's performance and quality against a collection of 1843 Sudokus.
  + I wrote all this in C++ intending to integrate it into the project with the JNI.  I quickly discovered it would be easier to port the code to Java instead, which I then did.  Predictably, performance suffered.
    - The Java version is slower by roughly a factor of 3 on average, a factor of 7 in the worst case.
    - That sounds bad, but on my laptop it solves most test puzzles in about half a millisecond because DLX is just really, really fast.
  + The repository contains the solver in both languages and the Java version is included in the project.
  + I also managed the repository and refactored some project code written by other team members.
* **Zachary Ganger – Co - GUI and Interface Designer**
  + As interface designer, I worked closely with Mark to tie together the solver, the randomizer, and the GUI.
  + My job was to make sure that once we figured out what the GUI looked like, it controlled things as they should work
  + I started by adding the “Solve” button to the keypad (later moved below the puzzle) and implementing the Solve code to solve some puzzles that we hard-coded for testing.
  + After I was sure the solver worked, I implemented the random puzzle generator.
  + After that, I helped Mark to finish formatting the colors and layout of the GUI, including changing the board to be a square rather than a rectangle covering the whole screen, moving the solve button to the game page, and implementing a hint button by making an array holding the solution to compare with the original (“solve” actually just fills in the original rather than displaying a second solution array).
  + I was also responsible for much of the testing, using an emulator to run after each change was implemented and testing each feature of the app on each launch.
  + In addition, I added Toasts (pop-up messages) for errors, when the game cannot complete a command, or when the puzzle is finished.
  + My final contribution was to help Mark again with the addition of the help page. We had to add a second Activity, which we had not had to do throughout the rest of the project. However, with the addition of a separate “About” class and an XML layout file that corresponded with it, we managed to make a separate page for the game description.
* **Mark Laubender - Co - GUI and Interface Designer**
  + I'm responsible for the Sudoku.Java, Activity\_sudoku.xml, KeypadDiaolog.java, keypad.xml, and GameView.Java classes and files.
  + I was responsible for the implementation of all buttons and layouts in the user interface, as well as all dialog boxes.
  + In Sudoku.Java I was responsible for all onClickListeners, void onClick(View v), void startGame(int), void openRules(View), and void openNewGameDialog().
  + The GameView.java activity solely consists of drawing the board, and the game.java consists of the logic behind the game.
  + I was responsible for the buttons, layout and color, drawing the board (onDraw(canvas)), selecting a square and drawing a selection rectangle(onTouchEvent(MotionEvent), void select(int, int), void getRect()), and changing a selected tile number (void changeSelectedTileNumber(int)).
  + I was responsible for the button and the function path leading from the solver to the gameboard (instantiating a solver and calling solvesudoku()).
  + I was responsible for all buttons, onClickListeners, and layouts. I'm responsible for the hint button (void solver() which calls gameView.hint(), which returns one integer for the specified x and y location).
  + I was also responsible for interfacing the GUI, the random generator and the solver together.  Creating function paths from both the random generator and the solver to the user interface.
* **Julie Frish – Documentation Manager**
  + During group meetings, I took notes on what each person had done during the week as well as what is still needed to be done. Any suggestions and plans are also included in meeting notes.
  + I offered moral support and snacks for extended group meetings when finalizing the project
  + The icons needed for the app were created in GIMP by me.
  + The next blockbuster short film about Sudoku327 was shot, directed, and edited by me. I also star as 3 of the main characters (myself, freaky goth chick, and super nerd).
  + All documentation was handled by me. I put together the Project Documentation which outlines the front- and back-end of the project. I created the “Read Me” with instructions on how to play the game. Additionally, this beautiful Statement of Work was compiled by me (pun intended).