Design Document for Programming Project 3

Section 0 – Introduction:

Development Team Members:

Azam Jamal Email: ajamal6@uic.edu
Zakee Jabbar Email: zjabba2@uic.edu
Ahmed Khan Email: akhan227@uic.edu

CS 342 – Project 3

Project Name: Sudoku Solver

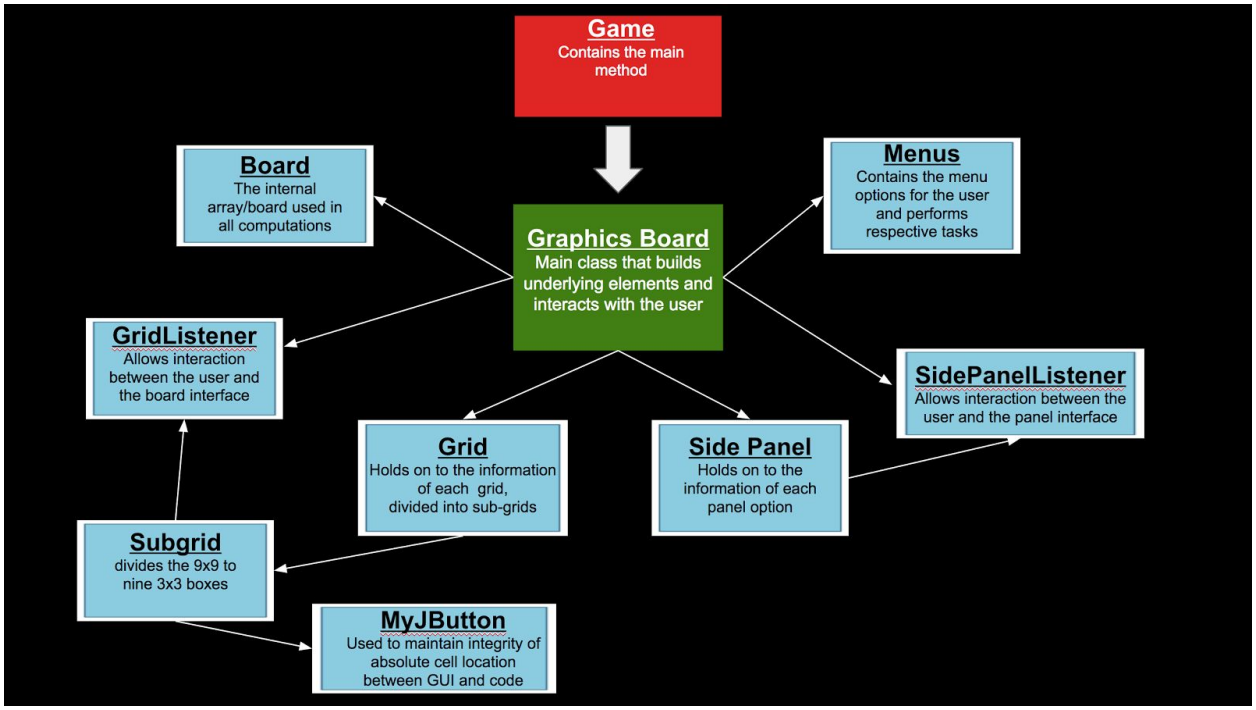
Section 1: State the purpose of your project/sub-system:

Sudoku is a popular puzzle that uses a 9x9 grid of 81 cells, divided into rows, columns in boxes. Each row, column and box contains nine cells, and the overall goal of the puzzle is to fill the numbers from 1 to 9 so that each row, column, and box contain each number contain a number from 1 to 9 only once.

The goal of this project was to write a solver that attempts to find a situation that will resolve (or help resolve) a value in a cell in a Sudoku puzzle. Our team had to write a GUI program in the Java programming language which allows a user to “play” a Sudoku puzzle that provides assistance to the user for the following situations: *Singles*, *Hidden Singles*, *Locked Candidates*, *Naked Pairs*.

The purpose of this project is to help assist Sudoku players in solving a Sudoku puzzle, in any given situation, and also helps them in strengthening their own Sudoku solving skills!

Section 2: Define the high level entities in your design:



Section 3: For each entity, define the low level design:

Board.java (Class)

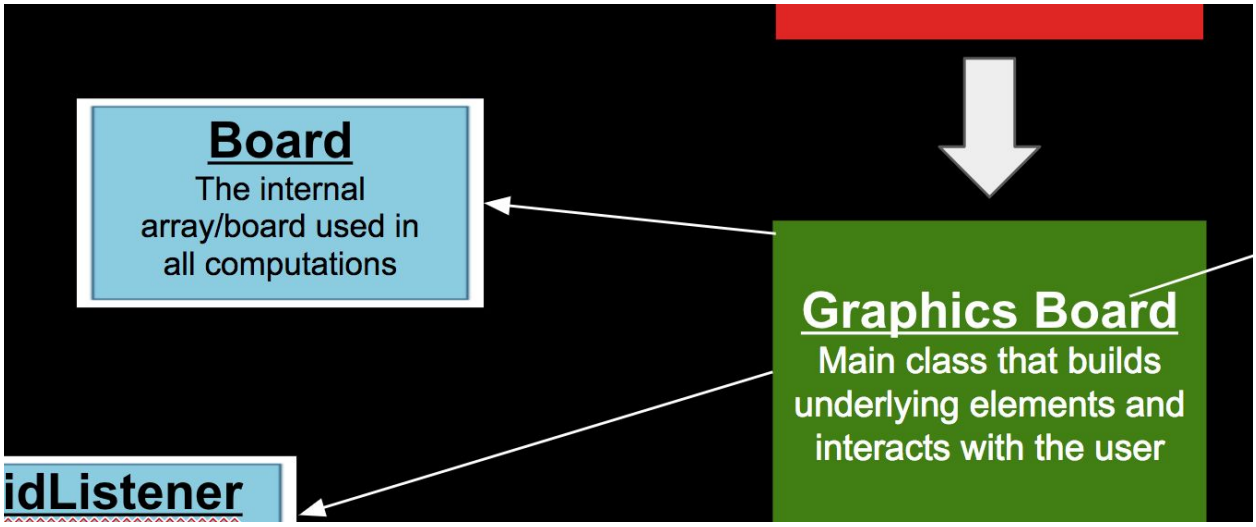
Usage:

This class is used to set up the internal board from the game, and interact with the graphics board. It sets and returns the value of a specific position, as well as returns the position of a specific value. The Board class also reads candidate lists, and updates and resets, or fills the board when necessary, as well as telling the user whether they won or not. As far as interacting with candidate lists, the main service the class provides is removing values of candidate lists for row and columns.

This class makes good use of modifiers and contains clean code. There are four ArrayLists implemented as the class' properties. There is an integer variable for the board, which is initialized as a 2D array. The ArrayList that stores a cell's candidate list is a 2D ArrayList. The reason we used ArrayLists is because ArrayList is variable length, which means ArrayList can

grow or shrink in size dynamically, which is beneficial. ArrayLists also don't require an initial size.

Model



The Board Class is basically the internal board (in form of array) used in all computations. It interacts with the Graphics Board, which is the main class that builds the underlying elements and interacts with the user.

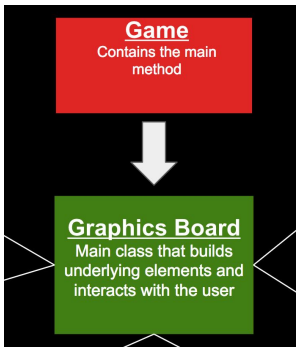
Game.java (Class)

Usage:

This class just only contains one function, the main method of the program. It creates a Graphics Board class, which when executed, displays the board, and then the user can proceed playing Sudoku from there. In other words, it executes the game. This simple class is very clean since it only contains one function, and it is easier how the classes interact with each other this way.

Benefits: Easy to read, clean
Risks: none

Model



The Game class is the class that contains the main method. That is what executes. It also interacts with the Graphics Board class.

-

GraphicsBoard.java (Class)

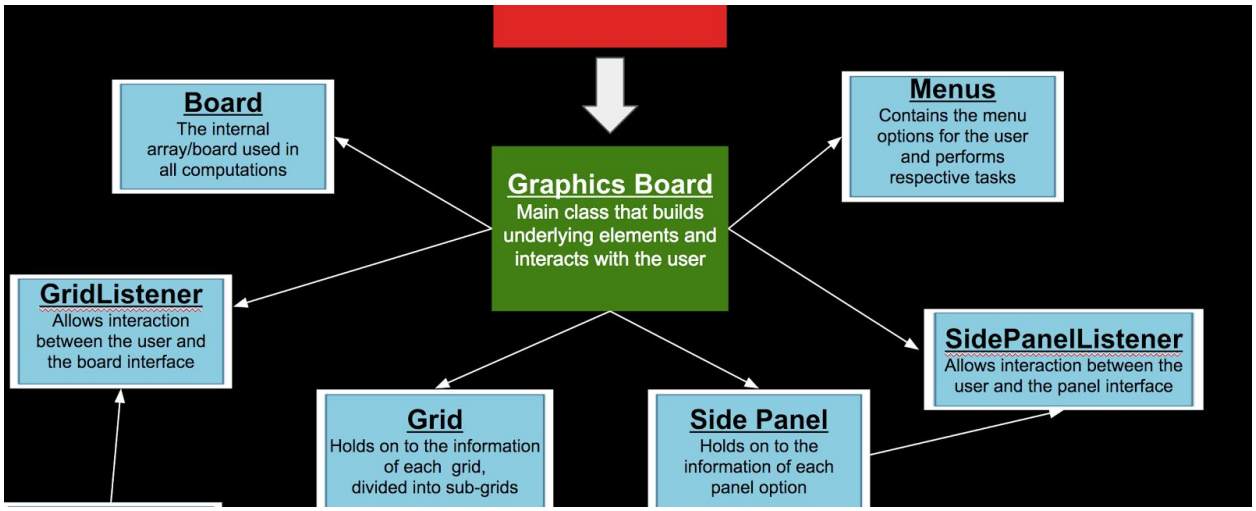
Usage:

The purpose of this class is to create the game board and also using event handling code to deal with clicks of the user. We use encapsulation in this method, with many set methods. Encapsulation provides value to this class as it promotes maintenance and code changes can be made independently. Some set methods include: setting current value, setting clear mode, setting “Check Fill” mode, etc. Other methods include whether it’s in candidate mode, clear mode, etc. The toughest part of making this class was writing the code for the constructor, but the rest of the class includes some getter methods, along with setter methods that deals with the backend of the Sudoku puzzle. The GUI code is the biggest part of this class, using JFrame, JLabel, JPanel, and dealing with user interaction.

Benefits:

-Encapsulation used to promote maintenance
-interacted well with GUI; everything is well-organized
-easy to understand what class does and what is going on
-most of GUI code is located in this class
-board looks visually appealing

Model:



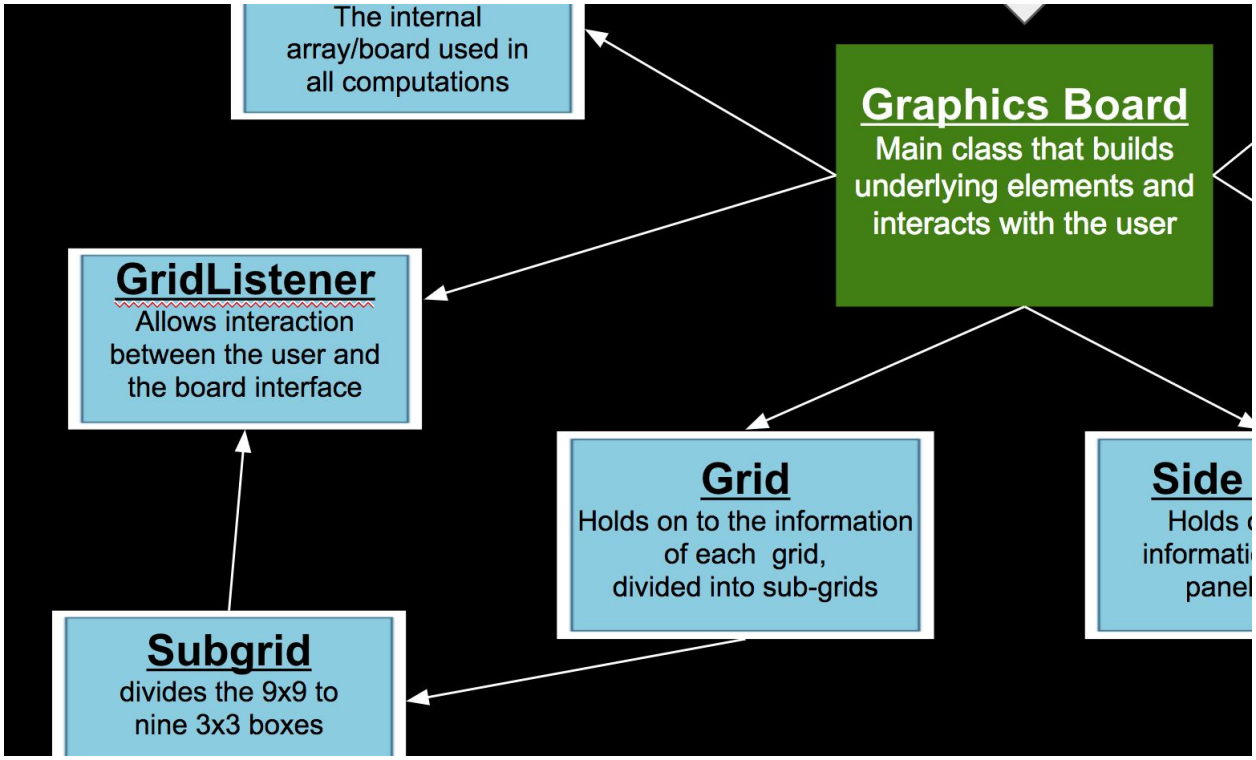
As you can see, the Graphics Board class interacts with many of the classes in this program. The Graphics Board class is the main class that builds the underlying elements and interacts with the user.

Grid.java (Class)

Usage:

This class sets up the subgrids and sets up some properties and attributes for the grid in the Sudoku puzzles. It sets up nine panels of type SubGrid, which is it's own different class, so there is interaction between these two classes. There is special method in our class where values are set, and many if-statement conditional statements are used to evaluate it. Even though it works fine, there are more efficient ways this could have been done. The other two methods used in this class is a method that acted as a listener for the panels, and there is also a method that resets the panels when appropriate.

Model:



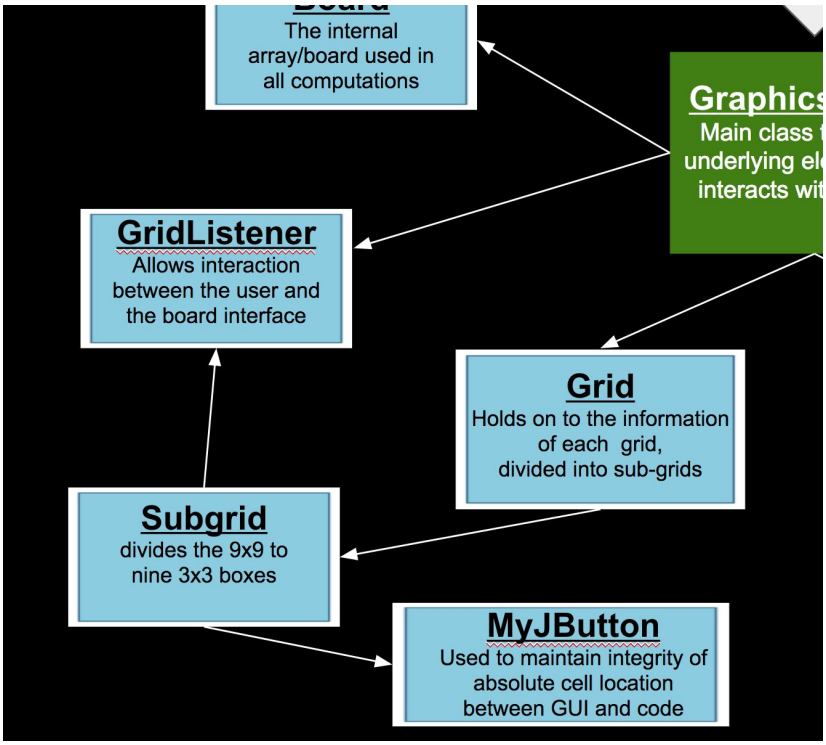
The Grid class holds onto the information of each grid, and it is divided into sub-grids. It interacts with Subgrid and Graphics Board.

GridListener.java

Usage:

Similar to the previous class in this list, we use event handling to interact with user and grid. It's used with the Grid class; the GUI deals with events. Based on a specific click of the mouse, the program takes specific action corresponding to the click.

Model:

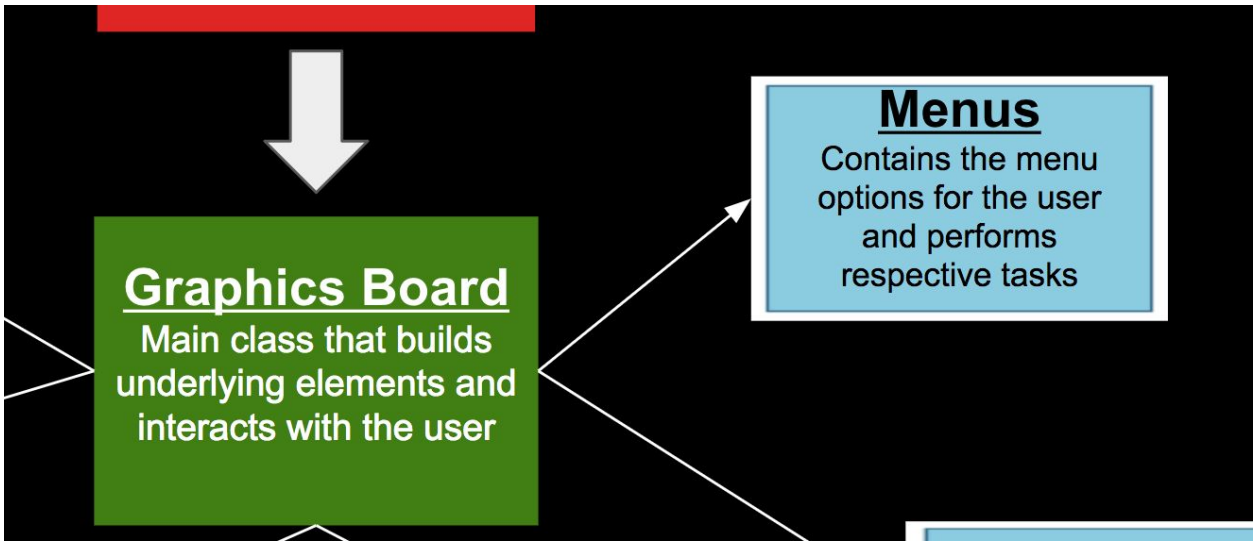


GridListener class allows interaction between user and board interface. It interacts with Graphics Board class as well as the Subgrid Class (which divides the 9x9 to nine 3x3 boxes).

Menus.java

Usage: This is one of the most important classes of the program. This class makes the menu for the game, and this is where the four situation algorithms come into play. While the actual algorithms are written in another class, this class listens to which option in the menu the user clicks on. It then performs that action, whether it is the 'About', 'How-To-Play', or invoking any of the four algorithms, auto-solve, and "Check-On-Fill Mode". It uses the actionPerformed() to perform that option. The constructor in this class allows you to open, load, or save file.

Model:



The Menus class contains the menu options for the user and performs respective tasks, and interacts with the Graphics Board class.

-

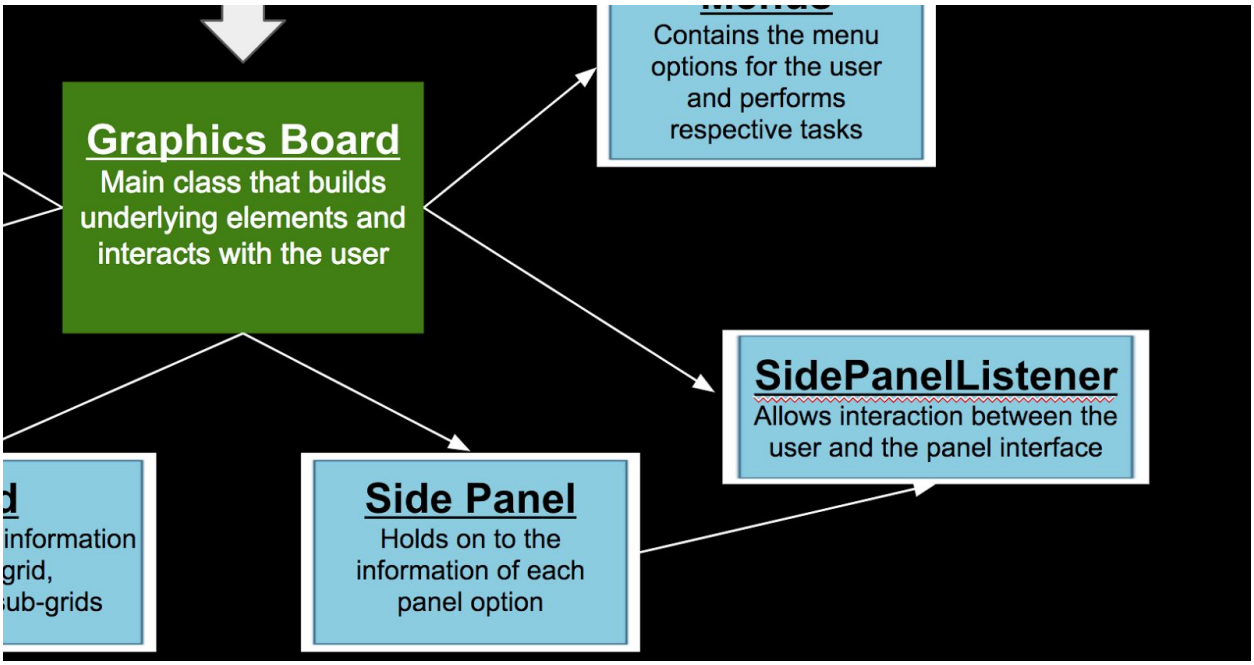
SidePanel.java

Usage: We divided each essential part of the board into their own classes. This class consists of setting the buttons and making them functional. We also added a listener for each button.

Benefits:

-each button has it's own listener, avoiding any event handling errors
-visually appealing, easy to click on

Model:



The Side Panel class holds onto the information of each panel option. It interacts with Graphics Board class and the SidePanelListener class.

SidePanelListener.java

Usage: Similar to the previous class in this list, we use event handling to interact with user and side panel. It's used with the SidePanel class; the GUI deals with events. Based on a specific click of the mouse, the program takes specific action corresponding to the click.

Model:

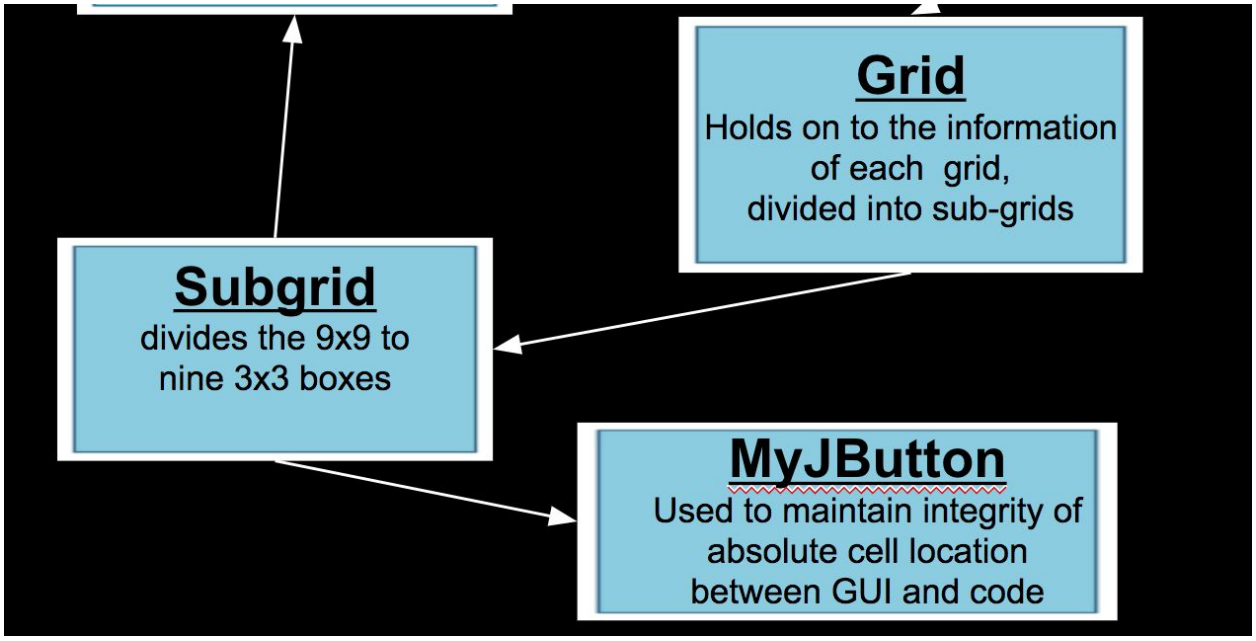
The SidePanelListener class allows interaction between user and panel interface. It interacts with the Side Panel class and Graphics Board class

---
-

SubGrid.java

Usage: This sets up subgrids for the grid for the Sudoku board. With our own implementation of the MyJButton class, that's how the buttons were made for the subgrid. A listener was also added for the subgrids.

Model:



The Subgrid class divides the 9x9 to nine 3x3 boxes. It interacts with the Grid class as well as the MyJButton class.

-

Coord.java

Usage: The purpose of this class is to store x and y coordinates of algorithms.

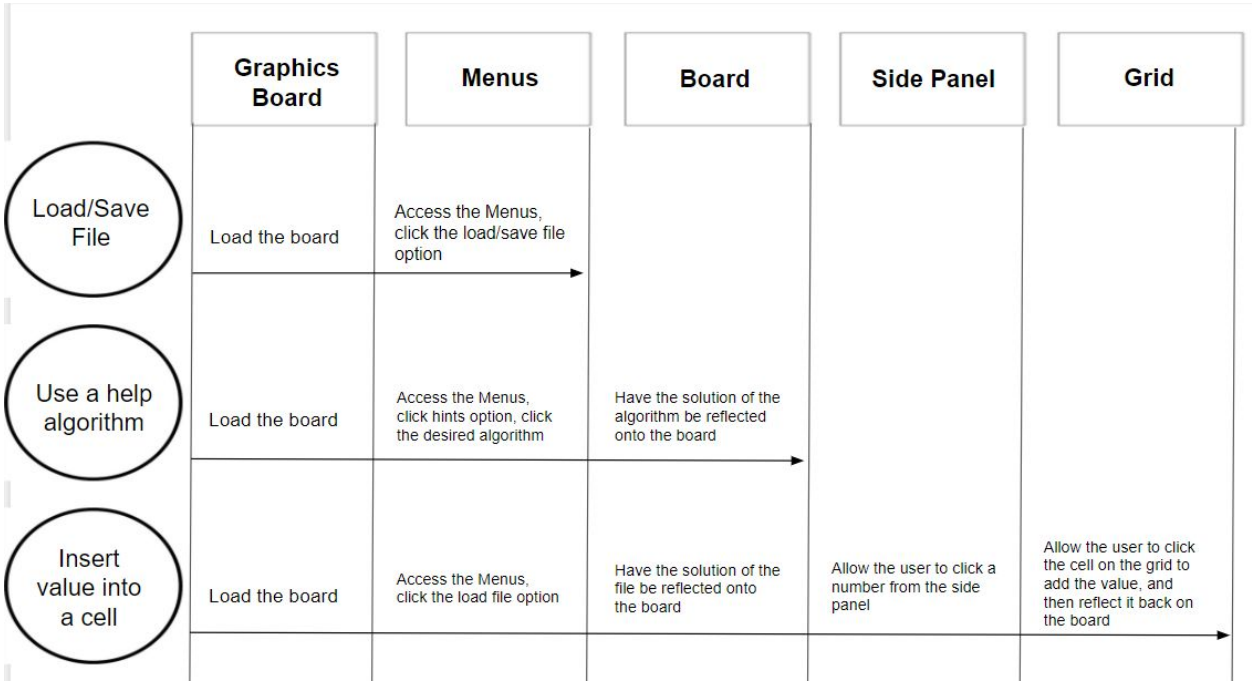
---

myJButton.java

Usage: An additional class to make our own “JButton” to store the x and y values each individual button corresponds to, as well as if the specific is able to be modified.

INTERACTION:

Interaction diagram:



Section 4 - Benefits, assumptions, risks/issues:

- Benefits:

1.) ArrayLists are used over lists, because ArrayList is variable length, which means Arraylist can grow or shrink in size dynamically, which is beneficial. ArrayLists also don't require an initial size.
2.) The use of encapsulation helps promote maintenance and code changes
3.) Each part of the board has it's own class that manages the back end of that specific part
4.) The design follows the standard of clean code, includes comments when appropriate, good variable and method names, methods are created and sued when needed, etc.
5.) Proper use of access modifiers

- Assumptions
    - We assume user will not run algorithms if candidate lists are destroyed by putting incorrect values

- Risks/Issues
    1.) While the use of ArrayList allowed the program to function properly, it would be better to use HashMap instead
    2.) Some algorithms could have had better efficiency.
    3.) The size of some files of the program are too big and have too much control.

Section 5 - Conclusion

This documents highlights the design we used for our project and highlights what classes we used as well as how everything (classes and objects) interacts with each other and the user. There were many benefits of our design as well, as well as a few risks and some things to improve on, however, the pros outweigh the cons!