Artificial Intelligence I - Homework 6

Participants:
Olivia Shouse
Ryan Christopher
Luke Schaefer

Language: Python

Important Information
- The rows and columns in our sudoku board are zero-indexed
- To run our program run the main.py. It will allow you to pick instances 1, 2, or 3. It will then print out the selected four variables, including their domain size, location, degree, and value placed. It also prints out the completed board and execution time.

Classes
- Board
    - Attributes
        - Board[row][col] - 2 dimensional array of boxes
        - OpenSpaces - Number of open boxes in board
        - RowDictionary - Dictionary containing values that are in each row
        - ColDictionary - Dictionary containing values that are in each col
        - GridDictionary - Dictionary containing values that are in each grid
            - Grids are the 9 3x3 array of boxes
    - Getters and Setters
    - PlaceMove
        - Adds the new move to the board and updates the affected domains and degrees
    - FindMove
        - Finds the best move to make based on MRV and Degree Heuristics
    - PrintBoard
        - Prints the current state of the board with the value from each box
    - SetDomain
        - Sets the domain of a given box in the board based on other boxes in its row, col, and grid
    - SetDegree
        - Sets the degree of a given box in the board based on other boxes in its row, col, and grid
    - UpdateDomainAndDegrees
        - Updates the domain and degree values for all boxes affected by a move
        - This contains the Forward Tracking Implementation
        - If it fails, it returns false
    - RemoveMove
        - Sets the current move to 0.

        - It updates the domains and the degrees of all the boxes affected in the row, column and box.
    - GetLowestDomain
        - Finds all the moves with the lowest domain and returns them in an array
    - GetHighestDegreeAfterMRV
        - Finds the move with the highest degree out of array of moves found after MRV has been run
- Box
    - Attributes
        - Value - Contains the value of the box
            - Initialized to 0 when the box is empty
        - locRow - Row location of box
        - locCol - Column location of box
        - locGrid - Grid location of box
        - Domain - All possible values that the box can be
        - Degree - Number of other boxes that are constrained by this box
            - Based off of all values in the domain
    - Getters and Setters
    - GetGridRange
        - Returns the row and column range for any given grid value
    - PrintBox
        - Prints out all the values in the box

Results

A.

- CSP**:**
    - Variables:
        - Each box on the grid, 81 in total
    - Domains:
        - {1, 2, 3, 4, 5, 6, 7, 8, 9}
        - The value that each box could potentially hold
    - Constraints:
        - Each row, column, and 3x3 grid must have all values 1-9 within it
        - 9 row constraints
        - 9 column constraints
        - 9 block constraints
- BackTracking Algorithm
    - Follows the backtracking algorithm that also uses MRV and Degree Heuristics
    - Uses recursion to continuously update the board with the most likely next move that is found based on MRV and Degree Heuristics
        - Attempts to place all values in the domain starting with the smallest value first

        - If move is placed and forward tracking passes, then the move is kept and the backtracking is called recursively
        - If the move is placed and forward tracking fails, then the move is removed and the next value in the domain is used
    - Backtracking algorithm continues to run until there are no longer any empty boxes within the board or the board is not solvable
    - Contains forward tracking by updating the domains and degrees that are affected by each move that is placed
- ForwardTracking Algorithm
    - Implemented in the board class with the function "UpdateDomainsAndDegrees"
    - This function updates all of the domains and degrees in the row, column, and block that are affected when the value is placed.
        - It does this by looking at the specific column, row, and box that are affected when the user places a move.
        - It calls helper functions setDomain() and setDegrees() to determine the updated domain and degree of each box in the affected column, row, and box.
- MRV
    - Our MRV functionality is used to find the empty boxes in the board that has the smallest domain
    - getLowestDomain() is a function on the Board class that scans through the board and returns an array of all the boxes with the smallest domain
- Degree Heuristics
    - Our degree heuristics functionality is used to find the box that has the highest degree.
    - getHighestDegreeAfterMRV() is used to find the remaining boxes that have the highest degree of the results of the MRV function
- Tiebreaking
    - tieBreak() is used to find the box that has the smallest col value and smallest row value out of an array of boxes
    - This is used when the results of MRV and Degree Heuristics contain more than one box

B.

Instance 1

	Variable Selected	Domain Size	Degree	Value
Variable-Value Assignment 1	Row: 4, Col: 5	1	8	7
Variable-Value Assignment 2	Row: 4, Col: 2	1	11	2
Variable-Value Assignment 3	Row: 4, Col: 4	1	10	5
Variable-Value Assignment 4	Row: 4, Col: 1	1	12	9

Instance 2

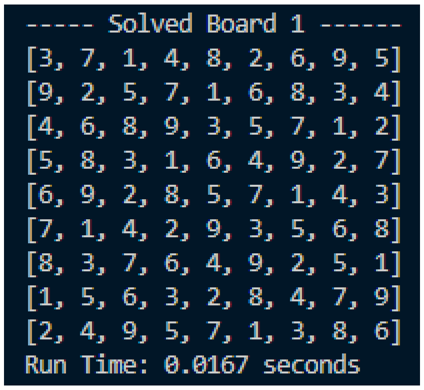
	Variable Selected	Domain Size	Degree	Value
Variable-Value Assignment 1	Row: 1, Col: 1	2	12	7
Variable-Value Assignment 2	Row: 0, Col: 1	2	13	3
Variable-Value Assignment 3	Row: 2, Col: 1	1	12	8
Variable-Value Assignment 4	Row: 5, Col: 1	1	12	9

Instance 3

	Variable Selected	Domain Size	Degree	Value
Variable-Value Assignment 1	Row: 1, Col: 0	2	13	4
Variable-Value Assignment 2	Row: 2, Col: 0	1	10	8
Variable-Value Assignment 3	Row: 3, Col: 3	2	13	1
Variable-Value Assignment 4	Row: 0, Col: 2	2	12	1

C.

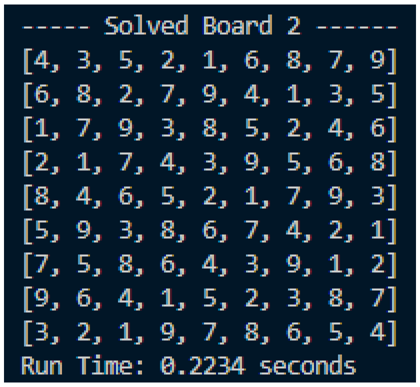
Instance 1



```
----- Solved Board 1 ------
[3, 7, 1, 4, 8, 2, 6, 9, 5]
[9, 2, 5, 7, 1, 6, 8, 3, 4]
[4, 6, 8, 9, 3, 5, 7, 1, 2]
[5, 8, 3, 1, 6, 4, 9, 2, 7]
[6, 9, 2, 8, 5, 7, 1, 4, 3]
[7, 1, 4, 2, 9, 3, 5, 6, 8]
[8, 3, 7, 6, 4, 9, 2, 5, 1]
[1, 5, 6, 3, 2, 8, 4, 7, 9]
[2, 4, 9, 5, 7, 1, 3, 8, 6]
Run Time: 0.0167 seconds
```

Execution Time: 0.0167 Seconds

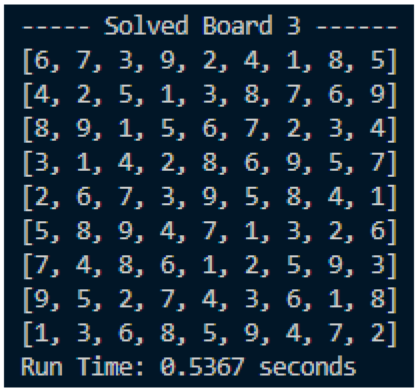
Instance 2



```
----- Solved Board 2 ------
[4, 3, 5, 2, 1, 6, 8, 7, 9]
[6, 8, 2, 7, 9, 4, 1, 3, 5]
[1, 7, 9, 3, 8, 5, 2, 4, 6]
[2, 1, 7, 4, 3, 9, 5, 6, 8]
[8, 4, 6, 5, 2, 1, 7, 9, 3]
[5, 9, 3, 8, 6, 7, 4, 2, 1]
[7, 5, 8, 6, 4, 3, 9, 1, 2]
[9, 6, 4, 1, 5, 2, 3, 8, 7]
[3, 2, 1, 9, 7, 8, 6, 5, 4]
Run Time: 0.2234 seconds
```

Execution Time: 0.2234 Seconds

Instance 3



```
----- Solved Board 3 ------
[6, 7, 3, 9, 2, 4, 1, 8, 5]
[4, 2, 5, 1, 3, 8, 7, 6, 9]
[8, 9, 1, 5, 6, 7, 2, 3, 4]
[3, 1, 4, 2, 8, 6, 9, 5, 7]
[2, 6, 7, 3, 9, 5, 8, 4, 1]
[5, 8, 9, 4, 7, 1, 3, 2, 6]
[7, 4, 8, 6, 1, 2, 5, 9, 3]
[9, 5, 2, 7, 4, 3, 6, 1, 8]
[1, 3, 6, 8, 5, 9, 4, 7, 2]
Run Time: 0.5367 seconds
```

Execution Time: 0.5367 Seconds