Table of Contents

Search Algorithm.java

The main class for the project.
This class calls the FileReader class and contains the base of the CSP.

```
public static void main(String[] args) {
```

This function is where the project starts. The FileReader class and the cspAlg function is called (if the input file is not found, then a relevant exception is thrown and the program is stopped). This function also controls the timeout. The project is set with a 30-second timeout; however, it can be modified by altering the variable named "timeout" (time is in seconds).

```
public static int cspAlg(char[][] tiles, int[] targets, int[]
tileCount, int totalTiles)
```

This function contains the CSP. The tile array and the first node is initialized, then there is a backtracking algorithm that uses relevant heuristics and calls the AC3 algorithm.

```
static ArrayList<Node> getNeighbors(Node current)
```

This function creates an arraylist of valid neighbors to the current node and returns the list to cspAlg. If a neighbor has a layout value or target value that exceeds the goal, it will not be added to the list.

```
static void solutionPrint(Node current)
```

This function prints the final tile list with the layouts that satisfy the goal. The uncovered bushes can also be printed by uncommenting lines 128-132. For assurance, the current layout count and bush color count is printed - the values should match the goals.

```
static boolean inOpen(Node node, ArrayList<Node> openList)
```

This function checks if the given node is in the openList, which is an array list that contains all unevaluated nodes.

```
static boolean inClosed(Node node, ArrayList<Node> closedList)
```

This function checks if the given node is in the closedList, which is an array list that contains all evaluated nodes.

Arc.java

An arc is the relationship between two nodes.
This class constructs arcs for use in the AC3 algorithm.

```
public Arc(Node one, Node two)
```

Constructor for an arc. Node one is the neighbor, while node two is the current node or the parent node.

```
public Node left()
```

Returns node one (the neighbor).

```
public Node right()
```

Returns node two (the parent or current node).

```
public static Arc arcCreate(Node n)
```

Returns an arc created from a neighbor (n) and the parent node.

```
public boolean equals(Object other)
```

If two arcs are equal, it will return true.

```
public String toString()
```

To print the arc in a digestible form. For simplicity, it prints the current target numbers.

ConstraintProp.java

This class is focused on constraint propagation. This is where the AC3 algorithm is housed.

```
public boolean AC3(ArrayList<Node> neighbors)
```

The AC3 algorithm that checks arc consistency. Returns true is consistent, false if not.

```
public static boolean arcConsistency(Node node)
```

This function is called in the AC3 to ensure all arcs are consistent. Since the domain is dependent on the number of each layout, an arc is considered inconsistent if the current layout or target values are less than the parent (or greater than depending on which layout).

FileReader.java

A class to read in the input text file and organize the data into usable forms.

```
public void readFile(String name) throws Exception
```

The base function of the class. This function reads the file as an input stream into a string list, then calls the other class functions to retrieve important sections of the file.

```
public static int dimensionCalc(List<String> text)
```

Calculates the dimensions of the landscape by counting the number of lines before reaching the curly bracket of the layout goals. All comment lines and empty lines are ignored upon reading the file in.

```
public static int findTotalTiles(int[] tileCount)
```

Calculates the total number of tiles by using the goal number of tiles given in the file.

```
public static void landscapeArray(int dimension, char[][]
landscape, List<String> text, int totalTiles, char[][] tiles)
```

This function creates the landscape as an array organized by rows, then uses the array to create the 4x4 tiles in the correct order.

```
public static int[] setTileCount(int size, List<String> text)
```

This function creates an integer array of the goal tile numbers. The function uses characters that are only found in a certain layout name to determine the order in which they are organized:

- OUTER_BOUNDARY = ‘T’
- **F**ULL_BLOCK = ‘F’
- EL_**S**HAPE = ‘S’

Depending on if the relevant letter appears before or after the others, the following number is assigned to the organization chosen for the project (0 = Outer, 1 = El, 2 = Full).

```
public static int[] setTargets(int size, List<String> text)
```

This function creates an integer array of the goal bush color numbers.

(1(0), 1(2), 2(3), 3(4))

```
public int[] getTargets()
```

This function returns the target numbers. It is used in SearchAlgorithm.

```
public char[][] getTiles()
```

This function returns the landscape tile array. It is used in SearchAlgorithm.

```
public int[] getTileCount()
```

This function returns the goal tile numbers. It is used in SearchAlgorithm.

```
public int getTotalTiles()
```

This function returns the total number of tiles. It is used in SearchAlgorithm.

Heuristic.java

This function houses the heuristics used in the Algorithm.

```
public static int mrvCalc(ArrayList<Node> openList)
```

The minimum remaining values - if the remaining target distance (difference between goal and current state) is greater than the current min, the min is updated. The smallest difference is chosen.

```
public static int lcvCalc(Node neighbor, int index)
```

The least constraining values - this function tries all layouts for a certain tile of a node to determine which layout leaves the greatest distance between goal and current state. The number for the layout option is returned and changed within the getNeighbor function of SearchAlgorithm.

```
public static int TIE(int currentIndex, int currentMin,
ArrayList<Node> openList)
```

The tie breaker function for MRV - uses the layout distance to determine which option to use.

Layouts.java

A class that holds the layout organization.

```
public static char[] getEl(int option)
```

This function holds four el layouts for each of the rotations. The integer ‘option’ is used to assign the relevant layout needed.

```
public static char[] getFull()
```

This function holds the full layout.

```
public static char[] getOuter()
```

This function holds the outer layout.

```
public static char[] getInitialLayout()
```

This function holds the initial layout. It is the full layout; however, a separate function is made in case changes need to be made.

Node.java - Tile

Holds the constructor for the tiles.

```
public Tile(char[] val, char[] lay, String name)
```

Constructor for a tile. The character array ‘val’ contains the bush color values, while ‘lay’ contains the layout. The string ‘name’ is the matching layout name.

```
public String toString()
```

To print the tile in a digestible form. Prints the value array, layout array, and string name.

Node.java - Node

```
public Node(Tile[] tile, int[] targetNum, int[] tileCount)
```

Constructor for a node. The variable for the project.

```
public Node(Node base)
```

Constructor for a node based on another node’s data.

```
public int[] layoutNumberCalc()
```

Counts how many of each layout is currently assigned.

```
public int[] targetNumberCalc()
```

Counts how many of each bush color is currently assigned.

```
public boolean finalCheck()
```

This function checks the current layout and bush color number. If the number equals the goal, then the solution is found.

```
public boolean layoutCheck()
```

This function checks if the layout option is within bounds. Removes domain if not.

```
public boolean targetCheck()
```

This function checks if the bush color numbers are within bounds.

```
public boolean changeLayout(int tile, int option, int el)
```

This function changes the layout of the relevant tile and updates the necessary values.

```
public void distCalc(Node node)
```

This function calculates the difference (distance) from current values to the goal values and updates the current node.

```
public int[][] tileOrder()
```

This function organizes the tile array with the order of tile with least number of values to the tile with the most. The actual tile array remains unaffected, but a secondary multidimensional array containing the order and uncovered values is created

```
public boolean equals(Object o)
```

To test if two nodes are equivalent.

```
public String toString()
```

To print the node in a digestible form. Prints the all tiles in numerical order, the current and goal layout values, the current and goal bush color values, the target difference, and the layout difference.