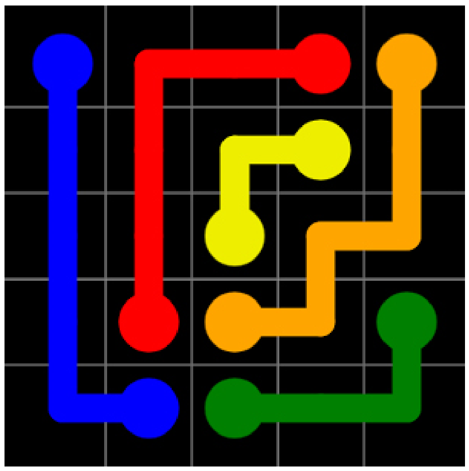
‘FLOW’ SOLVER

FINAL PROJECT REPORT

CS 5100 – FOUNDATIONS OF ARTIFICIAL INTELLIGENCE
FALL 2015



ADITYA GHOSH

NIKHIL SREENIVASAMURTHY

POOJITHA KARLAPALEM

Northeastern University – December 2015

TABLE OF CONTENTS

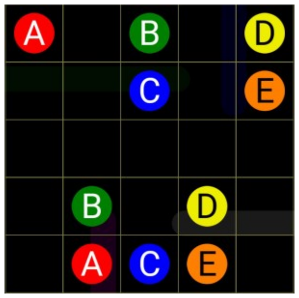
1. INTRODUCTION

Flow is a single-player game in which the player is given a grid (n x n) with a number of pairs of colored tiles.

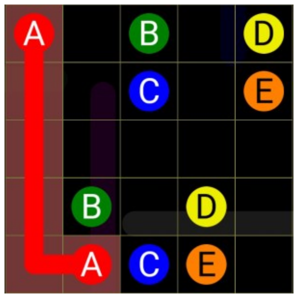
The rules of the game are as follows:

- The player must draw a path between the two given tiles of the same color for each color.
- The entire board must be filled to win the game.
- Paths may only connect vertically or horizontally, never diagonally.
- Only one color may exist on a given tile.
- The path for a color should be continuous.

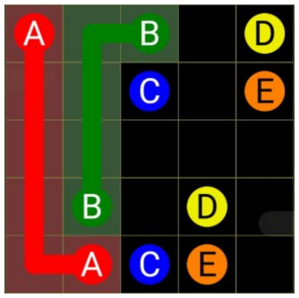
An example of the game play is shown below:



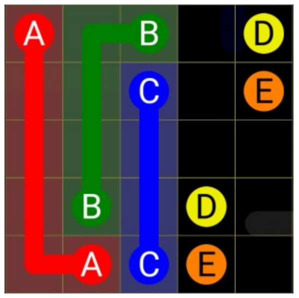
Initial Grid



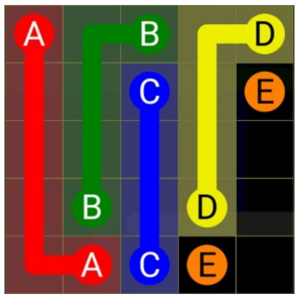
Step 1



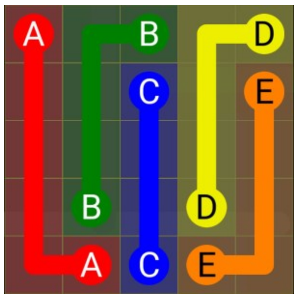
Step 2



Step 3



Step 4



Step 5

Follow the below link to play the web version of Flow
http://games.cdn.famobi.com/html5games/f/flow-free

2. PROBLEM DESCRIPTION

The grid is represented as an n x n array, where each cell [i , j] represents a color. A cell can be either empty or can contain only one color at any given time. A given grid is first assigned with all the start and end cell for each color whose positions are given as an input.

Assumptions:

a. *Each color has an initial and a final position described in the problem.*
b. *There exist at least one path from the initial to final position of each color.*
c. *There exists a solution to the given problem.*

Finding a solution to flow is an NP-Complete problem.
So, for very large grids, time taken by any algorithm to find a solution increases exponentially.

Proof: According to Wikipedia, Numberlink (a problem very similar to Flow) is NP-Complete.
Reference: "Kotsuma, Kouichi; Takenaga, Yasuhiko (March 2010), NP-Completeness and Enumeration of Number Link Puzzle, IEICE technical report. Theoretical foundations of Computing 109 (465): 1–7".
https://en.wikipedia.org/wiki/Numberlink

For the exact inputs and outputs, refer to the "TESTS and RESULTS" section of this report.

3. PROOF OF CONCEPT

The problem could be formulated this way:

- SEARCH: The solver uses search algorithms with backtracking to efficiently find a path between pair of positions for a color in the grid.
    - Backtracking: The search backtracks in case there is no path ahead.
- CONSISTENCY CHECK: We ensure that a path exists for every subsequent color that has not yet been assigned a path.
    - Backtracking: If the consistency check fails, backtracking is performed the most recently assigned color until consistency is ensured.
- The search is then repeated for each subsequent color until each cell of the grid is assigned with one color.

4. ALGORITHM

The Algorithm used to find a path for each color is a modified DFS algorithm. We have modified it to get a successor based on the direction of the final position of the color.
We have used DFS because a solution always exists at the bottom of the search tree.

```
solve(problem,visited=[])
  • grid = initializeGrid(problem)
  • currentColor = Pick a start color.
  • currentDirection = choose a random direction
  • while grid is not complete:
      o If successor exists in current direction:
          ▪ Move in the currentDirection
          ▪ Add currentPosition to visited
      o Else If successor exists in any other direction
          ▪ Change direction and continue
      o Else
          ▪ Backtrack until there exists a successor except for visited
          ▪ Change to that direction and continue
      o If currentPosition is final position for currentColor
          ▪ if all subsequent colors have a path to their final position
            (Path is found using DFS)
              • Move to next color and Continue
          ▪ Else
              • Backtrack until a path for each color is ensured.
              • Change direction so that the last position does not go the same way.
```

5. DATA STRUCTURES

Problem:
Problem is a dictionary with keys as colors and values as a list consisting initial and final coordinates for that particular key/color
Coordinates is a tuple of two numbers (x, y).
e.g.:

```
problem = {color1 : [(c1xi, c1yi), (c1xf, c1yf)],
           color2 : [(c2xi, c2yi), (c2xf, c2yf)],...}
```

Grid:
Grid is a class with members as size and problem as input parameters
Grid also internally maintains a gridDict to keep track of color assignments.
gridDict is a dictionary with keys as coordinates and values as either None or a color
e.g.:

```
grid = {(x1, y1) : color1, (x2, y2) : color2, (x3, y3) : None, (x4, y4) : color1, ...}
```

Direction:
Direction is a class consisting of NORTH, SOUTH, EAST, WEST. The definitions for the above are:

```
NORTH = ‘n’, SOUTH = ‘s’, EAST = ‘e’, WEST = ‘w’
```

Sub problem:
SubProblem is a class to represent the state of a single color with the grid at that instance as a reference. This is essential in finding existent paths for the remaining colors.

Stack:
Stack is a class maintaining a list as a member.
Stack offers the standard stack operations such as push and pop.

6. TESTS AND RESULTS

Run `Tests.py` *to view more problems and FlowSolver solutions.*

How to run the code:

```
grid = solve(size, problem)
grid.printGrid()
To get the traversed path for each color uncomment # print visited in solve function in
FlowSolver.py
```

Inputs:

```
problem4 = {"A":[(0, 0), (3, 3)], "B":[(2, 1), (1, 3)], "C":[(0, 3), (1, 2)]}

problem5 = {"A":[(0, 0) , (1, 4)], "B":[(2, 0) , (1, 3)], "C":[(2, 1) , (2, 4)],
"D":[(4, 0) , (3, 3)], "E":[(3, 4) , (4, 1)]}
```

To solve problem4, run:

```
grid = solve(4, problem4)
grid.printGrid()
```

Problem	Input Representation	Output
problem4	# [A] [ ] [ ] [ ] # [ ] [ ] [B] [ ] # [ ] [C] [ ] [ ] # [C] [B] [ ] [A]	# [A] [A] [A] [A] # [A] [A] [B] [A] # [C] [C] [B] [A] # [C] [B] [B] [A]
problem5	# [A] [ ] [B] [ ] [D] # [ ] [ ] [C] [ ] [E] # [ ] [ ] [ ] [ ] [ ] # [ ] [B] [ ] [D] [ ] # [ ] [A] [C] [E] [ ]	# [A] [B] [B] [D] [D] # [A] [B] [C] [D] [E] # [A] [B] [C] [D] [E] # [A] [B] [C] [D] [E] # [A] [A] [C] [E] [E]

7. SCOPE FOR IMPROVEMENT

This search could be made such that after completing one color, the consistency is checked for remaining colors such that each subsequent color has a unique path from its initial position to final position in the grid. Unique path here ensures that the intersection of positions for each path of remaining colors is empty.
The current implementation does not target a unique path, but only checks for a path, which might be coinciding for two or more colors.

The path for a color that fills in maximum number of cells without failing the consistency should be chosen. This would improve the number of backtracks required, hence improving the overall performance of the algorithm.

When does it fail?

- When a path for the current color is blocked by a path of some other previously completed color, this algorithm fails, as we always backtrack only to the most recently completed color.
- Sometimes, when the colors and directions are not chosen in a specific sequence for complicated problems (problem having many different solutions), this algorithm fails.
- For complicated paths, there might be thrashing or infinite loops.

All these scenarios can be solved using the above mentioned improvements.

8. REFERENCES

I. Recursive path finding in a dynamic maze with Modified Tremaux's Algorithm.
- *by Nien-Zheng, Yew, Kung-Ming, Tiong and Su-Ting, Yong*.

II. Artificial Intelligence – A Modern Approach
- *by Stuart Russel and Peter Norvig*.

III. All images used in this report have been taken from the Free Flow game