Title: Flow Free Game

Name(1): Kyle Tam CDF ID(1): tamkyle1
Role: problem encoding

Name(2): Derek Wong CDF ID(2): wongsi13
Role: experimental assessment

Type of project: Search

Project Motivation/Background

We are making an AI that finds solution to the game Flow Free. Flow free is a game really popular when smartphones just started to get popular in the late 2000s with over 100,000,000 installs in Android phones. It is a simple but popular game.

For each puzzle, you have to pair all the colors by connecting them with pipes, while they cannot overlap each other. For each pair of colors, there is one of them that can be moved, acting as a robot, while the other one cannot be moved, acting as a sink. Since this game is a 1-to-1 problem and it can be formulated into a search problem with all the components needed, including state space, initial state, action and costs, search is an appropriate method to solve it.

Methods

A object of class FlowFreeState represents a node in the state space. Each FlowFreeState f contains the following key attributes:

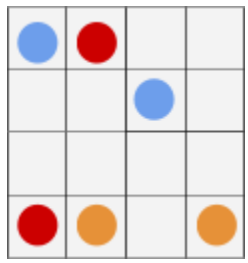
- f.action: the name of the action that generates this state from the parent (“START”, “UP”, “DOWN”, “RIGHT”, “LEFT”)
- f.gval: the cost of getting to the goal state
- f.parent: the parent state of f
- f.width: the width of the puzzle
- f.height: the height of the puzzle
- f.robots: a dictionary of all robots
- f.goals: a dictionary of all sinks
- f.paths: a dictionary of all walked paths
- f.filled: a set of indicies used

And for each FlowFreeState, it has a successors() function that returns a list of possible FlowFreeStates that can be generated from the current state through the actions.

Heuristic

We used 4 main tactics in the heuristics to help solve the problem.

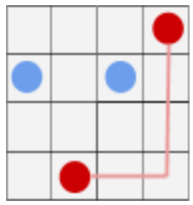
1. Calculating distance from robot to goal

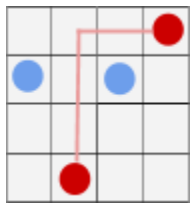


We based this heuristic on the Manhattan distance of each robot from its goal. We decided to make the AI solve pairs with shorter distance first because usually short paths shapes how the long paths should go most of the time (from experience of playing the game). Since we want to move the closest ones first we decided to apply some functions to the output of Manhattan distance so moving a closer robot closer awards more than moving a further robot closer. We wanted the function to be continuously increasing but increase slower as input gets bigger. The first thing that comes to our minds is log.

For example, for the puzzle on the right, if we link red first directly we will break blue but linking blue and yellow first will reveal a obvious path for red.

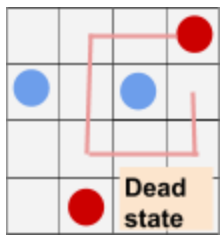
2. Prioritize robots walking on the side





Since using the sides as a path is usually the ones that makes the least conflicts with other paths we made a heuristic to incentivise robots staying on the side or walking on the side. Since the heuristic is “larger is worse” and we don’t really want to minus the heuristic number so we implemented this by adding slight penalties for other robots not on the side.

3. Ruling out dead state



We add super high heuristics for states that are dead states (states that will never turn into goal states). There are 3 dead states we considered: (1) when a robot is surrounded but not in goal (2) when goal is surrounded and no robot in it and (3) a robot in the wrong goal.

4- Solving robots 1 by 1

We give penalties for every robot not in goal state and a lesser but still significant penalty for every robot that moved. So robots will actually do path by path so conflicted states are exposed sooner.

Evaluation and Results

Test problems 0 - 17 are made to test certain aspects of the game or give AI specific challenges to see specifically what the AI is incapable of solving. Using heur_alternate & best-first search.

Problem	Dimension	Solved in 4 seconds	Node Expanded	Time Spent (s)	Cost
0	3x3	YES	15	0.01	4
1	3x3	YES	14	0.0	5
2	3x3	YES	14	0.0	6
3	5x5	YES	16	0.07	16
4	5x5	YES	108	0.01	8
5	5x5	YES	71	0.01	8
6	5x5	YES	944	0.16	13
7	5x5	YES	19	0.0	5
8	5x5	YES	589	0.09	13
9	5x5	YES	271	0.05	16
10	5x5	YES	1888	0.36	21
11	6x6	YES	679	0.13	21
12	6x6	YES	779	0.12	11
13	6x6	YES	2769	0.67	26
14	6x6	YES	1310	0.32	30
15	8x8	YES	2952	0.53	14
16	8x8	YES	8796	0.2	14
17	8x8	YES	1364	0.29	20

Test problems 18 - 34 are actual real challenging problems from the online version of the game. Testing how the AI fairs against “challenging for human” game problems. Using heur_alternate & best-first search.

18	5x5	YES	412	0.09	20
19	5x5	YES	250	0.078125	16
20	5x5	YES	1660	0.39	20
21	5x5	NO	9914	N/A	N/A
22	5x5	NO	8825	N/A	N/A
23	5x5	NO	9411	N/A	N/A
24	5x5	YES	484	0.171875	20
25	5x5	YES	429	0.140625	20
26	5x5	YES	1998	0.703125	20
27	5x5	YES	181	0.0625	18
28	6x6	YES	7542	0.31	30
29	6x6	YES	1108	1.08	30
30	6x6	NO	10211	N/A	N/A
31	6x6	NO	9282	N/A	N/A
32	7x7	NO	9862	N/A	N/A
33	7x7	NO	8475	N/A	N/A
34	7x7	NO	7318	N/A	N/A

Final result:
Our heuristic (heur_alternate) with best-first search:
27 of 35 problems (77.1428%) solved in less than 4 seconds.
Problems that remain unsolved in the set are Problems: [21, 22, 23, 30, 31, 32, 33, 34]

Manhattan distance heuristic (heur_manhattan_distance):
15 of 35 problems (42.857142857142854 %) solved in less than 4 seconds.
Problems that remain unsolved in the set are Problems: [8, 9, 10, 11, 13, 14, 18, 20, 21, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34]

Summary

From the final result, we can see that our heuristic improves a lot from the Manhattan distance heuristic. It solves 35% more problems. However, some complicated puzzles, especially the 6x6 and 7x7 ones, still remain unsolved.

Limitations/Obstacles

We tried to implement the mechanism of the original game - for each pair of colors, both points can be moved. However, while we were trying to make both points to be robots, there were too many states generated and it took lots of time for each problem to compute. Therefore, we restricted the number of robots to one and the results were obviously improved.

We tried a few different approaches to this problem. First we thought of it as a CSP with filling in every entry with different colour until the paths form but we quickly realize that approach is very impractical because the amount of constraints we need to make is insanely large. Then we thought of the Min-conflict algorithm approach suggested by the TA in the proposal response. We found that that approach is really similar to search. Because we found that there are too many ways to draw a path, we tried to find a shortest path for each pair first and than find another path if there is a conflict. We soon realized that we did not need to backtrack the whole path and just need to backtrack to the point before conflict. We realized that the algorithm has not much systematic difference with an actual incremental search. And we could make something similar to an incremental search by combining the search started code with a heuristic that force the system to workout path by path.

Conclusion

Throughout this project, we have learned how to turn a game into a search problem and implement search along with different heuristics. While analysing the results, we kept adjusting and improving our heuristic until a fairly decent result was obtained. Besides, we have learned how to reduce a problem to a more efficient one while the original properties are still retained. In the future, we would like to try to implement other strategies, algorithms and heuristics and see if there would be a better result for this game, and the more challenging puzzles can be solved where our strategy is incapable of.

Reference

[1]"Stream master,". [Online]. Available: https://games.cdn.famobi.com/html5games/f/flow-free/v4/?fg_domain=play.famobi.com&fg_aid=A1000-1&fg_uid=8557fc8f-26b3-4bc1-a770-d4fa798a30ca&fg_pid=4638e320-4444-4514-81c4-d80a8c662371&fg_beat=144.