CSL 356: Bubble Blast Game

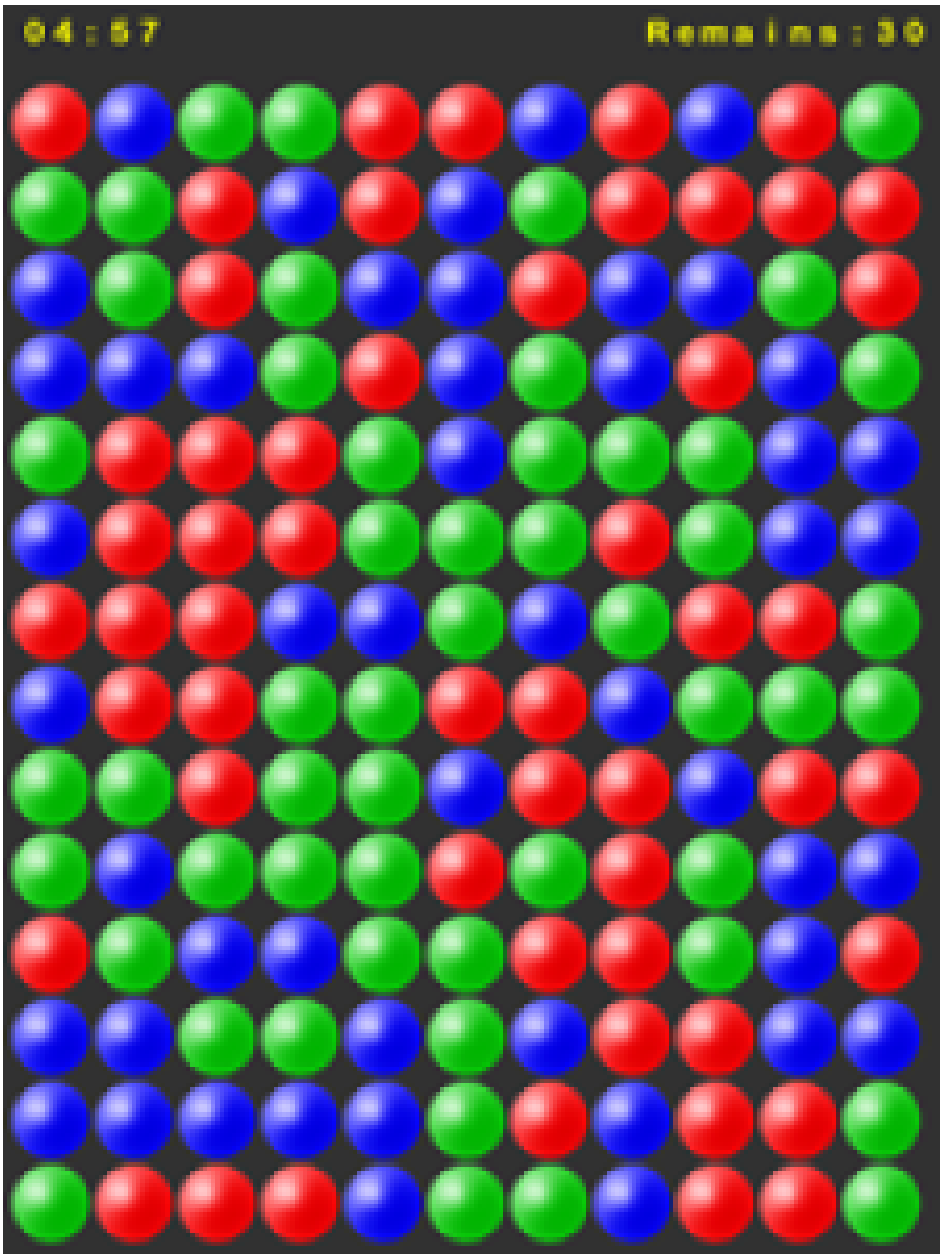
Due on Sunday, September 29, 2013

Sudarshan Iyengar

Jaskaran Singh

Contents

Problem 1



Best score for this grid- 3594(after 5 runs of the python script)

Problem 1 continued on next page...

Objective:To maximize the score of the popular mobile game BubbleBreaker.

Rules:
1)A valid move-Bubbles adjacent to each other of the same color have to be burst together.
2)You can't burst a single bubble.
3)Game ends when for every color the number of bubbles left is 0 or 1.

Input:An mXn matrix of numbers where every number has a value from 0 to c-1 where c denotes the number of different colors used.

Output:A list of states for obtaining the optimal solution from the starting configuration to the end of the game.
This list is [[M0,0],[M1,S1],[M2,S2],...,[MK,SK]] where **[Mi,Si]** is a list of 2 elements
Mi-Matrix of bubbles obtained after bursting bubbles from Mi-1 matrix
Si-Score obtained after blasting the bubbles from Mi-1 to get Mi matrix

Scoring:For every x bubbles burst in a move,score awarded is x^2

Problem 1 continued on next page...

Method:I have used randomization in solving this problem so as to get the optimal solution.
Possible moves:To find out what are the possible bubbles which can be burst I use Breadth-First-Search(4 neighbour connectivity) to find out all the connected components of the bubble grid.(Connected component would be those bubbles which are adjacent to each other and have the same color value.).
Move:A move consists of choosing a component at random from the bubble grid with number of elements in that component>1 and bursting it
Functions used:
Solver3:Function which solves the given grid by randomly picking the component to be burst till the game ends. I call the Solver3 700 times to get the best score out of all the attempts.
bfs:For finding the connected components of the grid so as to find all the possible moves to choose from. Returns a list of the connected components of the grid.
burst:Bursts the selected component putting a -1 in place of those bubbles in the matrix.Then it shifts down the columns of the matrix from which the bursting took place and moves all the -1s up.
After this we check that if a column is empty i.e full of -1s then we shift the entire submatrix before this column to the right
effectively removing the colum of -1s.In the end whatever deleted bubble(-1) we have, a colored bubble will lie only to the right
or below that deleted bubble.
Analysis:
I have experimented with different approaches before finally deciding the randomized version of the algorithm for solving the problem.
Some of the earlier approaches I considered were:
Greedy approach:
1)Largest block burst:
Implemented the largest block burst approach ie the component with the largest size is burst first. This I have implemented in the current code(Solver2) itself but I've commented it since I didn't find it optimal. A simple counter example where it doesn't give an optimal answer is:
xxxxbb
bbbbrr
where r is red,b is blue and x is empty.
If we consider this as the start stage and go to the end of the game the score obtained would be $4^2 + 2^2 + 2^2 = 24$
However,the optimal solution is $2^2 + 6^2 = 40$ ie burst the 2 red first.
2)Burst the lower blocks:
A counter example:
rrrr
yrry
gbbg
According to this approach we will first burst the 2b block,then the 4r block comes down and we burst it, then the 2g block is burst,then the 2y finally the remaining 2r block.Score=$2^2 + 4^2 + 2^2 + 2^2 + 2^2 = 32$
The optimal solution.score=$6^2 + 2^2 + 2^2 + 2^2 = 48$
Brute Force:
Exhaustive recursion:
Implemented a recursive approach(Solver1 is commented) which bursts a component and then chooses the next component to burst recursively.
This is a brute force approach and is the slowest and takes up a lot of memory on the stack because of the depth of recursion for large grids.Thus, its an infeasible approach for large matrices(m*n>30).

Problem 1 continued on next page...

Randomized approach
I finally implemented a randomized approach for solving the problem.
Proof:
Let the probability of finding the most optimal solution for a single run of the algorithm be s(which is extremely low).Then $f = 1 - s$ is the probability of failure to find the most optimal solution in that turn. To increase the probability of finding the optimal solution we have to perform this experiment a large number of times(k) to obtain a significant proability:
Prob(Success)=$s+fs+f^2*s+f^3*s.....+f^{k-1}*s \Rightarrow Prob(Success) = s*(1+f+f^2+f^3...f^(k-1))=(1-f^k)$
Thus to increase Prob(Success) f^k has to be decreased and it decreases with larger k.

Current Research paper-"The Complexity of Clickomania" by Erik Demaine
According to the paper titled "The Complexity of Clickomania"(Clickomania is the alternate name of BubbleBreaker)
the goal of the game is to remove all of the blocks, or to remove as many blocks as possible.
Formally, the basic decision question is whether a given puzzle is solvable: can all blocks of the puzzle be removed?
More generally, the algorithmic problem is to find the maximum number of blocks that can be removed from a given puzzle.
We call these problems the decision and optimization versions of Clickomania repectively.
According to the paper,
the one-column optimization version can be solved in $O(k * n^5)$ time.
the one-column two-color decision version can be solved in O(n) time.
the one-column two-color optimization version can be solved in $O(n^3)$.