Ryan Knowles
3-27-2014
MTH 5050 - Parallel Processes
Final Project Proposal

Problem Statement:
Procedural Content Generation is a field of study primarily within the game industry. The basic premise is to use algorithms to cheaply generate an endless supply of new content for the players. I will be analysing one method in terrain generation to generate natural feeling cave systems/tunnels using cellular automata.

Algorithm:
The following cellular automata algorithm is based on the one found at Reference 1.

I intend to extend this to work in three dimensions and optimize the algorithm for better performance via double buffering (this would remove the need to create a new matrix every iteration). I am also looking at ways to streamline the counting of a cell's neighbors.

```
public boolean[][] generateMap(){
    //Create a new map
    boolean[][] cellmap = new boolean[width][height];
    //Set up the map with random values
    cellmap = initialiseMap(cellmap);
    //And now run the simulation for a set number of steps
    for(int i=0; i<numberOfSteps; i++){
        cellmap = doSimulationStep(cellmap);
    }
}

public boolean[][] doSimulationStep(boolean[][] oldMap){
    boolean[][] newMap = new boolean[width][height];
    //Loop over each row and column of the map
    for(int x=0; x<oldMap.length; x++){
        for(int y=0; y<oldMap[0].length; y++){
            int nbs = countAliveNeighbours(oldMap, x, y);
            //The new value is based on our simulation rules
            //First, if a cell is alive but has too few neighbours, kill it.
            if(oldMap[x][y]){
                if(nbs < deathLimit){
                    newMap[x][y] = false;
                }
                else{
                    newMap[x][y] = true;
                }
            } //Otherwise, if the cell is dead now, check if it has the right number of
neighbours to be 'born'
            else{
                if(nbs > birthLimit){
                    newMap[x][y] = true;
                }
                else{
                    newMap[x][y] = false;
                }
            }
        }
    }
    return newMap;
}

//Returns the number of cells in a ring around (x,y) that are alive.
public countAliveNeighbours(boolean[][] map, int x, int y){
    int count = 0;
    for(int i=-1; i<2; i++){
        for(int j=-1; j<2; j++){
            int neighbour_x = x+i;
            int neighbour_y = y+j;
            //If we're looking at the middle point
            if(i == 0 && j == 0){
                //Do nothing, we don't want to add ourselves in!
            }
            //In case the index we're looking at it off the edge of the map
            else if(neighbour_x < 0 || neighbour_y < 0 || neighbour_x >= map.length ||
neighbour_y >= map[0].length){
                count = count + 1;
            }
            //Otherwise, a normal check of the neighbour
            else if(map[neighour_x][neighbour_y]){
                count = count + 1;
            }
        }
    }
}
```

Approach:
This algorithm is very similar to the Jacobi algorithm and so I intend to perform a similar time estimate on the code. I plan to parallelize the calculation of the new matrix from the old matrix using Open MP, similar to the Jacobi examples in class. I intend to capture the output of each iteration of the algorithm and transcribe it into either the json or csv format so that I can use it to create a time-lapsed visualization of the data space.

References:

1. http://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664
2. http://www.roguebasin.com/index.php?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels
3. http://pixelenvy.ca/wa/ca_cave.html
4. http://csharpcodewhisperer.blogspot.com/2013/07/Rouge-like-dungeon-generation.html
5. http://pcg.wikidot.com/pcg-algorithm:cellular-automata