A Technique for Random Map-Structure Generation for Strategy Games Project Proposal for Zachary Sparks

James Koppel

April 27, 2011

1 Introduction

Suppose you were to create a map for a game such as Starcraft by randomly picking the contents of each tile. Even limiting to those maps where there were exactly two starting positions, the resulting map would likely not lead to a very fun game.

This document describes a general program for creating random proto-maps, which are turned into final game maps by other programs. Discussion of this program is limited to creating proto-maps for a vaguely defined class of strategy games which I call closed-map extract-and-hire army strategy games, discussed below.

2 Goals and General Notes

To some extent, this project idea is intended to address how to get someone with little knowledge of Heroes of Might and Magic to write an important component of a random map generator for Heroes of Might and Magic II. I will discuss potential applicability to other games, but that is secondary. However, this is nonetheless an interesting approach to the problem of random map generation in its own right.

The goal of the implementation is to create something which can yield a variety of two-player maps which do not have glaring balance issues but yet are more interesting than symmetric maps, and which can be largely written

within 6 hours. This should really be thought of as a prototype, but not as a quick-and-dirty prototype. Extensibility is key – if used, expect various parts to be rewritten and optimized, heuristics to be added, parameters to be tweaked, interfaces to other systems set up, etc. A final version would need to allow parameters to be passed in. A prototype version would too, except that the manner of "passing them in" might just mean declaring them somewhere separate.

Implementation priority should be to create working versions of each section. If estimating the chances of winning a game of Force takes longer than expected to implement, finish that part, and omit the hill-climbing search and grid-mapping.

3 The Basic Idea

This problem can be formulated as an optimization problem. The idea is to randomly create a graph which abstractly represents the "areas" of a map and how they are connected with some information about how it will be turned into an actual map, and then perform a hill-climbing search to make it better under some value function. The program will also check that there is some way to render this graph into a proto-map. A proto-map is a way of partitioning a grid (for simplicity, stick to rectangular grids) of tiles into "areas," and annotating areas with information such as how many resources they contain. A final map generator will use this as a guide to create an actual map.

The value function is how equal both player's chances of winning are in an abstract Risk-like strategy game played on this graph. The next section describes this game, called Force.

4 The Game of Force

Force is a two-player, turn-based, perfect-information game played on a connected, simple graph. Every node is marked with an integer, called the node's "resource." A subset of nodes are marked as "force generators." A state of Force consists of an assignment from every node of the graph to the following:

1. A color, Red, Blue, or Gray, called the node's owner.

2. A nonnegative integer, the amount of "force" on that node, or the number of "force counters"

The two players are called Red and Blue. Red is first to move. Gray is neutral, and the amount of force on a gray node is also called "resistance." Gray nodes merely sit there until conquered by Red or Blue.

Each player's turn consists of three phases:

1. Moving: For each force counter, the player may move it to an adjacent node.

2. Resolving Combat: Suppose the player moves n force to a node of a different color which contains m force. If $n = m$, all force counters are removed, and the node does not change color. If $m > n$, then the node is "conquered" by the player and changes color to tha of the player. $f(m, n)$ force will now be on the node, where f is defined below. If $m < n$, the node will not change color, and $f(n, m)$ force will be left on the node.

3. Recruiting: The player computes r, the sum of the resource across all nodes of his color, and then places r force counters on a force generator of his color.

f should be a parameter, but I suggest $f(m, n) = \text{round}(m - \frac{n^2}{m})$. That is, if the winning player has k times as much force as the losing player, he should lose a fraction of $\frac{1}{k^2}$ of his force. This is taken from my observations from playing a variety of strategy games that the amount of losses taken by a stronger army decreases greatly as the ratio of army strengths increases.

Once all nodes on the board are one color, that player wins.

As an example game, consider the 3-node graph complete graph where all nodes have 1 resource and are force-generators, node 1 is initially Red with 1 force, node 2 is initially Blue with 1 force, and node 3 is initially Gray with 0 force. On Red's turn, he moves his 1 force to node 3, conquering the node, and then places 2 force on node 1. Blue player does nothing and places 1 force on node 2. Red then moves all his force to node 2, and conquers it, having 3 forces to Blue's 2, and thereby wins the game.

There are many possible variations on this game. For real-time strategy games, it may be a better approximation to play a version of Force with simultaneous moves, and where players cannot instantly hire force.. Don't worry about these.

5 Computing the Chances of Winning Force

I recommend sampling across many games played using Monte-Carlo game tree search, perhaps with some limitations to the considered move set such as rarely splitting up large piles of force. You may want to strongly bias against moves which split large piles of force. I do not really know what heuristic function to use, though one simple one is to let the "strength" of each player be the sum of their force and resource (with some sort of discount on the resource), and let the heuristic function be the difference or quotient of both players' strengths.

Allow a parameter function limiting the length of games in terms of the size of the graph. On a two-node graph with equal starting conditions, optimal play gives an eternal standoff.

6 The Ideas Behind Force

This is the part where I talk about all the layers and layers of ways in which Force is a poor approximation to the strategy games for which it is analyzing maps, and then turn around and claim its use should still be suitable. As a computer scientist, you should be used to this

Designing Force was (or rather, is, considering the design has not been tested and thus may need to change) a balancing act between creating a game sophisticated enough to reflect how map structure influences gameplay in closed-map extract-and-hire army strategy games but simple enough so that estimating the chances of winning is somewhat tractable.

"Closed-map" means that game maps typically restrict movement heavily: two distance areas in the map are unlikely to have an unobstructed straight-line between them. This is true of Starcraft and Command and Conquer, where a typical map contains many rivers, cliffs, and walls which naturally divide the map into "areas." This is very true of Heroes of Might and Magic, where very frequently adjacent "areas" are joined only by a single tile width, which is frequently blocked by a neutral army. This is, however, not true of Civilization, whih typically consists of completely passable large continents.

"Closed-map" also has a second meaning: The area graph does not change. This approximately includes Age of Empires, as building walls effectively changes the topology for non-siege units, while cutting down forests actually changes the topology (though one can approximate the first as leav-

ing an army on that tile, and design maps where the second is a non-issue). This excludes Master of Orion: Stars are effectively split into areas based on how far the player can travel from the nearest colonized star, but this structure changes as the player advances in travel technology.

Force reflects closed-maps by using a static graph and limiting force counters to moving between adjacent vertices. Note that this is effectively ignoring flying units, teleporters, and disallowing the creation of water maps.

"Extract-and-hire" refers to the mechanic of extracting resources from fixed locations on the map and using it to hire troops. This again includes Starcraft, where players are dependent on fixed locations of minerals and Vespene gas, and can build troops anywhere they have buildings. This includes Heroes of Might and Magic, where players draw resources from fixed locations of mines and towns. This excludes Supreme Commander, where players can eventually build power plants and mass fabricators that allow them to create exponentially-growing economies completely independently of the mass deposits spread around the map (I have won many games by turtling in this fashion). This, of course, also excludes Civilization and other 4X games.

The resource and hiring model of Force was designed to reflect this. I initially considered allowing for different types of resources and a schedule indicating how the amount of force a player can hire varies with the resources acquired (to e.g.: reflect how, in Heroes of Might and Magic, army strength takes a large leap once a player can begin producing the strongest creatures), but decided that was too complex (and would, hence, slow the speed Monte Carlo search converged). Force generators correspond to the places a player can hire creatures; in Starcraft, this would be anywhere; while, in Battle for Wesnoth and Heroes of Might and Magic, these would only be a few places.

"Army" simply means that the core game mechanic is of moving an army, with each unit independently mobile. This is true of all RTSs, as well as Battle for Wesnoth. It is only approximately true for Heroes of Might and Magic; armies can be split, but, for most of the series, each army must be led by a hero. However, skilled players typically concentrate almost all troops in one or two heroes, which indicates that splitting trooops will typically not lead to optimal branches.

The movement, combat, and win condition of Force reflects this.

There are many limitations to the Force model, such as (obviously) how all units are of a single type, which all have the same strength regardless of location. In more tactical games such as Advance Wars where being on

the right terrain tile is critical, this is a huge limitation; reifying an area remains a hard problem. Nonetheless, this does capture many more macro-level strategies such as making pushes to grab large resource deposits.

It's interesting how much an attempt to distill several strategy games down to the simplest elements that reflect their map structure produced something with such a strong resemblance to Risk; I did not notice this until writing it up. There are, of course, huge differences: Concentrating armies is much more effective, you can "abandon" territories by withdrawing troops, etc. Note that Risk is actually not an army strategy game, as you cannot freely move all armies during your turn.

7 Starting the Search

What makes a good starting is highly game-dependent, and thus you should not put much effort into this.

8 Hill Climbing

The allowable neighbors will, again, vary by the game, but will, in general, consist of removing and adding edges to nodes of low distance, and adjusting force, resource, and whether something is a force generator. Attempt to grid-reify a graph before simulating the game on it (see below).

Since the objective function is so heavyweight, some input from the game simulation will be helpful in making this converge to a usable map in reasonable time. For example, for whichever player has a chance of losing, you can find nodes likely to be captured by him in the early game and add resource or lower resistance.

9 Grid-reification

Parameters will be provided that allow you to compute a range on how many tiles a given node will take up. The task of grid-reification is to find a way to map areas into contiguous (convex?) sets of tiles such that nodes adjacent into graph are mapped into tile sets that share an edge. This is a constraint-satisfaction problem. Finding grid-reifications for modifications of graphs with grid-reifications should be easy, but I'm not sure how.

It may be a good idea to start the search by randomly creating a grid-reification, and then abstracting it to its graph, rather than randomly producing a graph and finding the grid-reification.

10 Previous work

Quite obviously, random map generation has been done before, though I have not found detailed discussion on other approaches.

US Patent $6,123,619$ is "Method of Generating Maps with Fixed and Random Portions and Use of Same in Video Games." I had trouble extracting any information from the writeup, other than an apparent emphasis on games with a single player-controlled character.

There is a fairly good whitepaper on terrain analysis for strategy games, with some discussion of applications to random map generation. It is also interesting for its discussion on partitioning a map into areas. http://www.gamasutra.com/gdcarchive

One of the first Google results for "random map generation" is for a fan-made RMG for Heroes of Might and Magic V. The creator summarized its workings as follows:

> The map generator uses the same algorithm as the HOMM3 RMG, expect it creates only two-player land maps (and lacks some of tables for the visual candy).
>
> Briefly, it works by first creating an abstract representation of the map and how the areas are connected (1). After that, the areas are created and filled with different terrains (2). In the last phase all the objects and guards are placed (3). 1.2.3.'