Procedural Content Generation in Strategy/Role-Playing Games

Nathaniel Buck

Submitted under the supervision of Professor Stephen J. Guy to the University Honors Program at the University of Minnesota-Twin Cities in partial fulfillment of the requirements for the degree of Bachelor of Science, *summa cum laude* in Computer Science.

[05/09/2013]

ABSTRACT

The world generation system for a turn-based strategy/role-playing game, codenamed "Project Death and Taxes," is implemented using a combination of a fractal-based algorithm and a Markov chain matrix process. From the generated world, a political map is created to outline macroscopic plot events for the game. A path-finding algorithm then computes a path between these political territories for the player to traverse, and discrete levels are determined along the path, with objectives and scenarios chosen stochastically. The existing system has some limitations; it is unable to recover effectively from failures in level generation and is unimplemented past level bounds placement, but it will be integrated into a more robust, complete system in the future.

TABLE OF CONTENTS

1. INTRODUCTION 4

2. PROCEDURAL CONTENT GENERATION 4

a. MOTIVATION 5

b. GOALS 5

c. FRACTAL-BASED METHODS 6

d. MARKOV CHAIN MATRICES 7

3. SYSTEM OVERVIEW 8

a. WORLD MAP GENERATION 9

b. POLITICAL MAP GENERATION 10

c. LEVEL GENERATION 11

d. SCENARIO GENERATION 14

4. RESULTS AND FUTURE PLANS 15

a. ANALYSIS 15

5. BIBLIOGRAPHY 15

INTRODUCTION

The video game industry is booming. Last decade saw the industry grow to over $10 billion in revenue, and it shows little sign of stopping (Entertainment Software Association). One recent trend in the industry, though, is a division between the highest and lowest echelons of developers: In the past, big-name, “triple-A” developers like Sony and Nintendo dominated, but smaller “double-A” developers, those who still employed hundreds but didn’t have quite the resources of the AAA teams, still thrived on good-but-not-great titles; nowadays, the AAA studios wield many millions of dollars in marketing funds to push high-selling, risk-averse titles (those from existing franchises, or those that conform well enough to genre conventions to suggest similar safety), while agile, independent development teams can create novel content at low overall cost, leaving the AA developers generally without a market (Capps 2012). All of these recent industry politics foreshadow a continued swelling of the “indie” development scene, and there has been growing prominence of tools to assist indie teams in rapid content generation to reduce production costs.

PROCEDURAL CONTENT GENERATION

This thesis focuses on the procedural generation of game levels for a tactical strategy/RPG, using methods to produce both the individual levels and the overall world map in which those levels are embedded. Terrain generation in particular is growing more common, and while the exact algorithms are often application-dependent, there are several methods that are frequently used. The approach presented here combines two methods of procedural generation: fractal-based generation and Markov chain methods.

MOTIVATION

Because of the high cost of hiring artists for a small development team, procedural content generation has received growing interest for new video games; why pay for an artist and a design lead when the game can generate levels, weapons, or characters on its own, leveraging the existing expertise of the game's programmers? There are myriad approaches to procedural content generation, and most systems use a composite of several algorithms. The game *Dwarf Fortress*, for instance, creates a massive world with simulated climate, geography, and historical events in a twenty-step process (Bay 12 Games). *Dwarf Fortress* has a single programmer, yet it has been heralded for its deep, complex environments, its vast opportunities for emergent behavior, and its ability to create gameplay experiences that feel "handcrafted and personal;" exemplifying the success of robust procedural generation systems (Weiner 2011). That need for "handcrafted" output is the fundamental challenge in designing such a system; since the purpose of procedural generation is to supplant the need for designers and artists, procedurally generated content ideally must match the caliber of human-created content.

GOALS

Since a content generation system must in some way match the quality of content produced manually, there need to be metrics to define how effectively that goal is met. A strategy/RPG is a game in which the player controls a group of unique characters through a series of distinct levels, each of which represents a single battle or encounter; as such, a generation system for a strategy/RPG must satisfy two goals: generate a set of discrete levels and apply some appropriate scenario and set of objectives to each level. When defining this in terms of quality of player experience, the metrics can be expanded: Considering some path through the game world that includes all levels, adjacent levels L_1 and L_2 should be chosen such that

$$Dist(L_1, L_2) < d \quad (1)$$

where Dist() gives the minimum distance between levels and d is some threshold. Enforcing this threshold ensures cohesion between subsequent levels. Similarly, for any two levels L_i and L_j,

$$Dist(L_i, L_j) \geq 0 \quad (2)$$

which restricts levels from overlapping, a convention that human-made levels would follow. Domain knowledge can be applied to determine more metrics: On average, strategy/RPGs consist of 25-30 levels, to provide about 30 hours of gameplay (Intelligent Systems, 2003). So,

$$25 = N_{min} \leq N \leq N_{max} = 30 \quad (3)$$

with N as the level count. In addition, variety is expected between different levels, so similarly,

$$6 = S_{min} \leq S \leq S_{max} = 10 \quad (4)$$

for S as the number of unique scenarios present across all levels; these values are chosen based on prior familiarity with similar games. Each level, then, should be designed appropriately for its scenario, with valid terrain composition and level dimensions.

FRACTAL-BASED METHODS

Fractal-based approaches are a fast, efficient way to generate terrain procedurally. In general, a fractal-based algorithm begins with a large shape and iteratively divides each of its edges in half, resulting in exponentially finer resolution of shapes, eventually leading to a complete loss of distinguishability between adjacent shapes. The diamond-square algorithm is a common, albeit old, approach: Beginning with a square with a height value at each of the four corners, at each edge the midpoint is inserted with the average value of the two points defining that edge. A point is also added at the midpoint of the square, which splits the original square into four. This is repeated iteratively, alternating between diamonds and squares with every

iteration (Fournier 1982). By perturbing the height values at new points, a randomized height map results that depends only on the initial values given; if the perturbation is scaled according to the iteration number, realistic formations occur, as earlier steps of the algorithm that affect larger areas of the overall shape have greater influence over the height values than do later steps which affect a greater quantity of values. Although the algorithm's running cost grows exponentially with each iteration, it produces exponentially many subdivisions, so it needs only to be run a number of times logarithmic to the desired resolution.

MARKOV CHAIN MATRICES

Markov chain matrix processes are not as robust as fractal algorithms but are more flexible and more general. As with all Markov decision processes, a state space is constructed with transition probabilities between states, such that the next state at any iteration depends only on the current state. For terrain generation, the process must expand outward from a seed point, so each state regards the terrain type assigned to a particular tile, which depends on the assignment of terrain types of its examined neighbors. The N terrain types in general form an N-clique, so any type could transition to any other with some probability; this allows the construction of an $O(N^2)$ table of the current tile's neighboring terrain types and the probability of assigning each terrain type, which represents the entire state space. To redefine the transition space then requires only using a different table. Also, each tile need only query the table with a random number and its neighboring tile types to determine its own tile type, and then recurse to unexamined neighbors. This proceeds in O(A * B * N) time, where A and B are the dimensions of the map to assign terrain types to, as each table lookup must query up to all N terrain types

with the selected random number and at each tile a constant number of operations is performed. While this can be costly for large A and B, for small subsets of a map it runs very quickly.

SYSTEM OVERVIEW

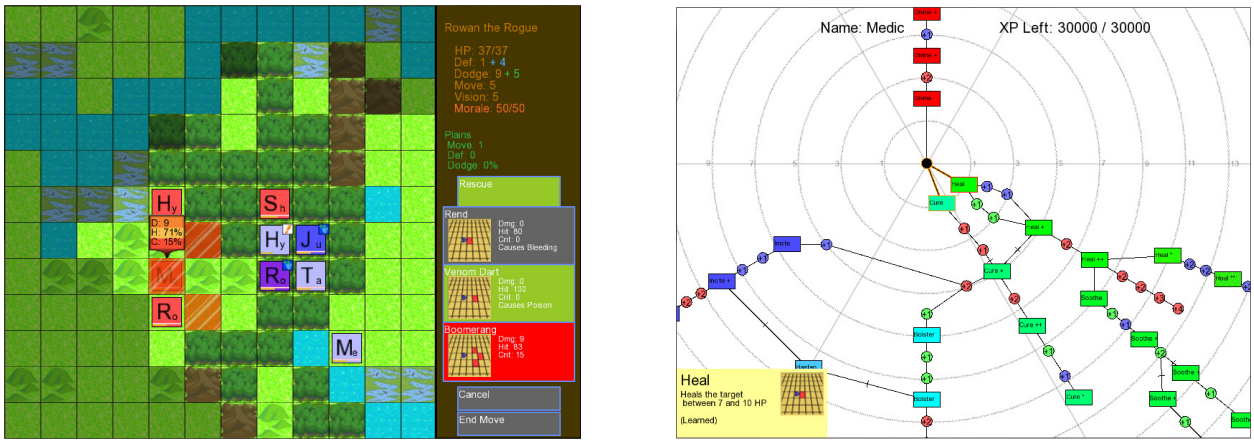

Figure 1: (Left) Prototype of PDT gameplay; (Right) prototype of character ability tree

"Project Death and Taxes" (hereafter PDT) is a turn-based strategy/RPG with a unique character development system that does away with typical class-based or item-based progression, giving each character its own unique abilities and attacks. Prototypes of multiple systems for the game are shown in Figure 1. In addition, the entire world in which the game occurs is procedurally generated, and the story is designed so that, apart from a handful of critical plot events, story elements can occur in arbitrary order. While the terrain generation used in PDT is based on existing techniques, the process of discretizing the game world into individual levels is an interesting problem; while generating each level independently is valid, creating a cohesive world that unifies the game's levels could yield a more engaging experience for the player. Combined with a story that is generally not pre-defined, PDT should be appealing for the extreme variety offered between multiple playthroughs of the game. This result is achieved through a multi-step process of random world generation, similar to games like *Dwarf Fortress*.

The overall nine-step process for terrain and level generation in PDT is outlined in Figure 2. First, the world terrain map is generated and used to create a political map to define the territories of each faction for story purposes. The map is then decomposed into individual levels, and then each level is assigned a scenario based on its content. The levels are then populated with content and objectives that correlate with their scenarios. These substeps are discussed in detail below.

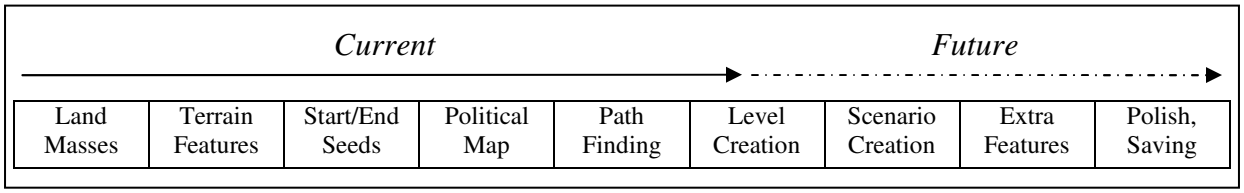

Figure 2: Overview of the world generation system in PDT

WORLD MAP GENERATION

The land masses for the world are generated using the diamond-square algorithm to create a height map, the values of which are then mapped to the terrain tiles used in the game (for instance, high values to Mountains and low values to Oceans). Onto this world is applied a series of Markov chain matrix processes to generate additional terrain features; for each of these, a seed point is randomly sampled that, if valid, begins a Markov process outward, with termination as a potential state to transition to at each iteration as a means of restricting the size of resultant feature. These Markov processes generate localized forests, marshes, and rivers for the map, although river generation could be handled during the height map generation (Smelik 14-15). After introducing a random number of valid terrain features of each type, start and end seeds are chosen, which serve as markers for the initial and final levels of the game. While a variety of requirements were considered for these seeds, such as connectivity along land masses, most seemed too prescriptive of valid plot events; why shouldn't the player, for instance, need to

charter a boat to cross from the first level to the next, instead of forcing the levels to be connected by a land path? As a result, only two restrictions are imposed: minimum distance, which is enforced by only sampling seed points from an ellipse near the map perimeter, and terrain type, to aid the first and last levels in conforming to the few predetermined story events. Figure 3 illustrates example output of these processes.

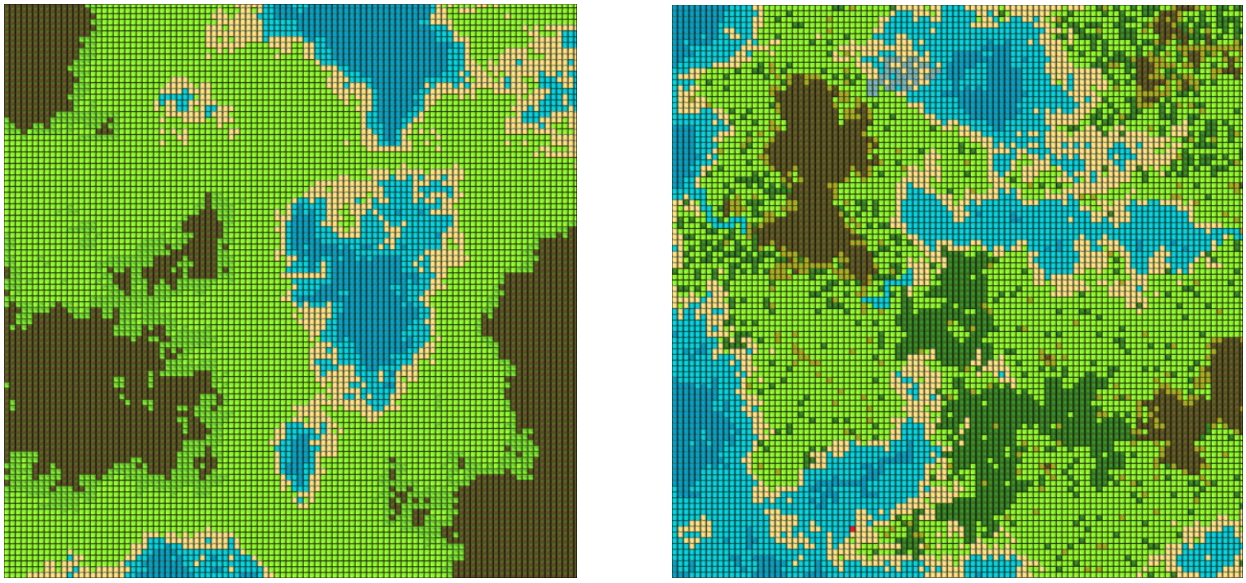

Figure 3: (Left) Subset of sample output of land mass generation; (Right) subset of sample output of Markov process generation with start seed placement shown in red (note that these are distinct subsets)

POLITICAL MAP GENERATION

A political map for the world is generated to provide guidance for the overall plot of the game: Each territory in the political map is seeded from a capital, two of which are the start and end seeds, two of which are chosen from terrain features generated by the Markov processes, and one of which is sampled randomly. All of these have a minimum distance enforced, though this is small enough that two capitals could have just one level in between them. After all capitals are chosen, an iterative, weighted, parallel grassfire expansion occurs: Each capital assigns its tile on the map to its faction and proceeds to the tile's neighbors, assigning them the same faction and

expanding to their unexamined neighbors, and so on until no tiles remain. This process weighs each faction based on how large its territory should be, so that one faction's territory could expand faster than another's, and at each iteration all factions are considered for expansion. An illustration is provided by Figure 4. However, the expansion considers tile type as it progresses, making terrain that is physically less passable also less passable politically; this sensibly shapes territories around major land masses and features and adds variability to the political map, when coupled with the randomness of seed placement. It could be (and has been in testing) that factions with the lowest weight control a plurality of the map, or it could be that they control tiny islands hardly large enough for a single level.

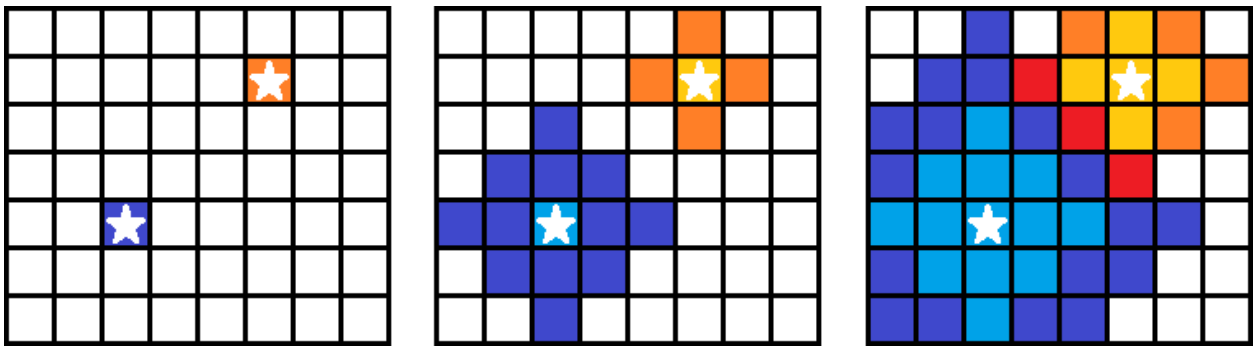

Figure 4: Illustration of territory expansion, with blue and orange "factions." Blue's weight is 2 to orange's 1. (Left) Both start at one seed. (Center) The state after one iteration. (Right) The state after two iterations. The assignment of the red tiles depends on which faction expands first. Note that this illustration lacks variable tile cost.

LEVEL GENERATION

Once both the physical and the political maps are determined, the map can be discretized into individual levels. The algorithm for this procedure has undergone several revisions, but at its foundation it consists of, beginning at the start seed, the expansion of the current level until it satisfies a set of conditions for some scenario, then an extension outward to define the starting point of the next level. This is a greedy algorithm, where the greedy choice is to place each level with the dimensions and location that first satisfy the constraints of any particular scenario. The

conditions for each scenario are generalized to allow all level bounds to be defined before any actual content needs to be generated, reduced to checks on the level's dimensions, terrain type composition, and how many instances of the scenario should be expected over the course of the game. For example, the first level will use the Tutorial scenario, of which there can only be one and it must be the first level, and it should consist of primarily Sand and Water tiles to ensure a coastline as its setting. This ensures that a generated level will always be appropriate for the scenario that it has been assigned.

The initial algorithm proved unsuccessful: A preliminary analysis indicated that creating levels in between major terrain features could be valid, since large bodies of mountains or lakes would generally not be desirable as the focus of an entire level but are still of particular interest. A raster scan across the map reveals such bodies, pausing at every unexamined Mountain or Water tile to determine the boundaries of its body and calculating the midpoint, with sufficiently large bodies broken into smaller components. Then, all rectangles of reasonable dimension (ratio of height to width, and height to width themselves) bounding pairs of the midpoints are amassed into a single list, with the intention of examining all rectangles to find those that are favorable for any particular scenario.

Whether or not this can be reasonably handled given the large quantity of potential levels, there exists still the problem of how to proceed from one level to the next, or how to order the levels once selected. To address this concern, the initial algorithm was replaced with one that creates each level individually in sequence, expanding out from one level to the next after the first is made fit enough for any scenario. To determine the direction of this expansion, an A* search is first performed between the capital seeds; the capitals are used for plot purposes, as major plot events will likely occur in capital cities regardless of world content. In testing, using

the nearest capital at each step did not yield a long enough path to be expected to satisfy Equation 3, but beginning with the path from the start seed to its second-nearest capital then back to the nearest one yields much more suitable paths (see Figure 5), although there tend to exist more collisions and overlaps along these paths because of the likelihood of backtracking. With a path defined, each level finds its boundary through which the path continues and samples a nearby point, and then extends from that point in the general direction of the path, accounting for terrain during the transition.

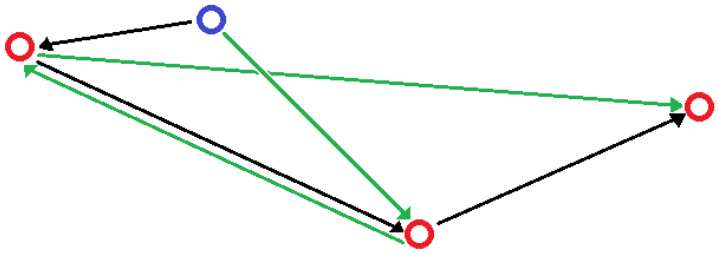

Figure 5: A simple depiction of the choice of nearest neighbor (black) and the choice of second-nearest neighbor (green) from the start seed (blue) to the other capitals (red); visibly, the green path is longer than the black path.

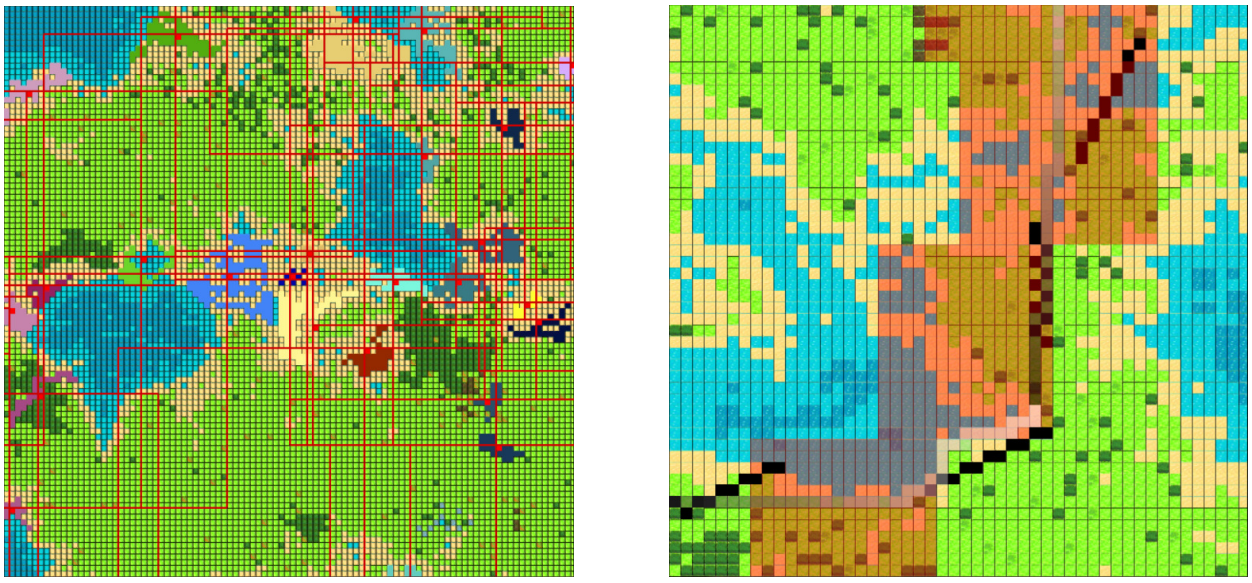

Figure 6: (Left) Initial level generation with bodies highlighted and potential levels outlined in red; (Right) current generation with levels in red, overall path in white, and expansion paths in black. Levels progress downward.

SCENARIO GENERATION

The existing system uses the above technique to generate the set of levels, accounting for multiple possible scenarios and avoiding overlapping levels when possible, although the heuristics for fitting scenarios are incomplete and its ability to handle cases in which no valid level is found needs further development. Once fully implemented, the system will proceed to populate each level with necessary constructs for its scenario, including enemy spawn points and buildings. Examining existing strategy/RPGs revealed a common approach in determining placement of these constructs: In general, the shortest path between the player start point and the objective is blocked by various obstacles, as in Figure 7 (Intelligent Systems, 2003). This "resistance" along the shortest path tends to yield interesting encounters elsewhere in the level, so it will be considered when placing these points. However, based on the results of the current algorithm, the amount of resistance should be partially considered during level construction, or else levels could be placed in locations with adequate composition to pass the preliminary check but containing terrain that does not afford this resistance. Once each level is populated, additional content is added, such as roads between towns created in different levels and small side levels for dialogue scenes to occur in, and the results of the entire process are saved so that the game may begin.

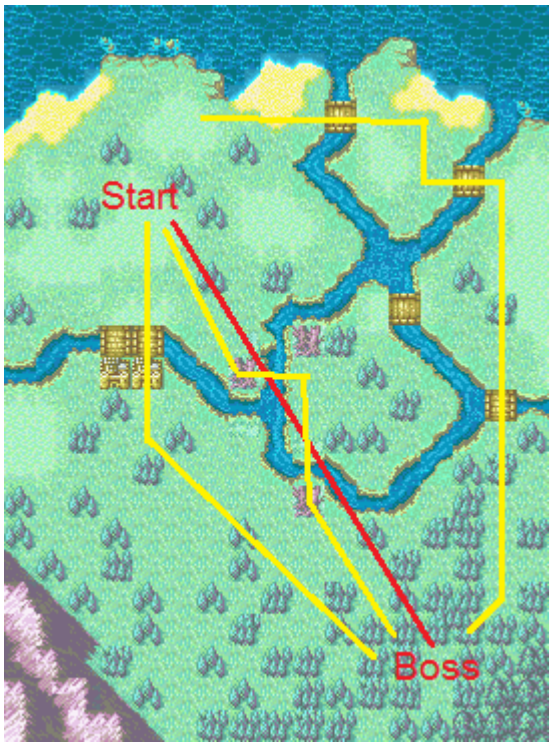

Figure 7: A level from Fire Emblem *(Intelligent Systems, 2003), with indication of straight path (red) to the objective versus actual paths (yellow)*

RESULTS AND FUTURE PLANS

As noted, the current system progresses no further than defining level bounds with simple rules enforced. This implementation has several limitations and is unable to complete the level set without heavy overlap, and it requires further refinement before the algorithm continues. Regarding the overall development of PDT, there are several other issues that need to be addressed: The initial prototype for PDT was developed as a fully playable multiplayer game using only Markov processes for map generation and with only one level generated, and the new generation system was built on top of this framework. This left it with several residual bugs, some of which proved too deep to allow robustness, and between those bugs and the repeated redefinition of the level expansion algorithm, not enough time remained to complete the system.

ANALYSIS

This implementation did show promise of the desired outcome: For the final game, 25 to 30 levels will need to be generated (as in Equation 3), each of which is expected to be 25x25 tiles on average. This requires, then, an absolute minimum path length of 625 tiles from the path-finding step, although in practice a minimum length of about 750 is more suitable. When testing the current implementation, a pre-loaded world is used with an A* path that is approximately 740 tiles long (an average length in the testing of many newly generated worlds), and the level generation results in 37±2 levels. Although there is significant overlap between some levels where the A* path crosses itself, the ~30% difference between the expectation and the actual result suggests that once overlap is completely disallowed Equation 3 will be satisfied, so the aforementioned use of second-nearest neighboring capital in path-finding produces reasonable paths. Only two scenarios were considered due to time, but the algorithm can include arbitrarily more scenarios, such that Equation 4 may be satisfied.

The system meets most of the other goals presented as well: Since levels are chosen only when they meet scenario constraints, they implicitly meet the needs for terrain content and dimensions for that scenario's objectives. The generated levels follow a consistent path through the game world with a maximum distance of 10 tiles between them as in Equation 1. They do not currently meet Equation 2 because of the necessity of level overlap, which is caused by the greedy level placement algorithm being caught in local minima; while the level placement will always satisfy a scenario individually, the fact that levels must adhere to the A* path means that placement currently cannot occur properly at path crossings, since positions of previously placed levels are fixed and could restrict the placement of future levels from ever satisfying a scenario without overlap. This can be remedied by allowing adjustment of level placement until all levels are generated, or by allowing a level to alter its initial position during generation on a failure.

Regardless, PDT will be overhauled in the near future to accommodate a stronger, test-driven framework, and the existing algorithm will be refactored into this framework. Changes to the level expansion will be great in order to refine level formation before the scenario generation phase, but the terrain generation will remain generally unchanged besides tuning. Additionally, all steps will be interruptible to reduce backtracking if any particular step fails; this will also allow multi-threading of generation to mitigate variably long generation times, always choosing the first world that completes its generation, and will allow initial world generation to begin in the background while the player handles other tasks like save file creation and introductory story cutscenes, reducing the perceived duration of world creation. These will not, though, supplant the entirety of the existing system; rather, the work completed for the current world generation will be augmented with better coding practices and structures, leveraging lessons learned from external projects to create a more robust, professional version of the system in the future.

BIBLIOGRAPHY

Bay 12 Games. *Dwarf Fortress*. Video game. N.p., 4 June 2012.

Capps, Mike. “Mike Capps on Finding the Next Cliff Bleszinski.” *Develop*. 20 Oct. 2011. Web. 28 Feb. 2013. < http://www.develop-online.net/features/1450/Mike-Capps-on-finding-the-next-Cliff-Bleszinski>.

Entertainment Software Association. “Essential Facts About the Computer and Video Game Industry.” ESA, 2010. Web. <http://theesa.com/facts/pdfs/ESA_Essential_Facts_2010.pdf>.

Fournier, Alain, Don Fussell, and Loren Carpenter. "Computer rendering of stochastic models". *Communications of the ACM* 25 (6): 371–384, June 1982.

Intelligent Systems. *Fire Emblem*. Video game. Nintendo, 2003.

Miller, Gavin S. P. "The definition and rendering of terrain maps". *ACM SIGGRAPH Computer Graphics* 20 (4): 39–48, Aug. 1986.

Smelik, Ruben Michaël. “A Declarative Approach to Procedural Gene ration of Virtual Worlds.” The Hague: TNO, 2011. Web. 2013. <http://graphics.tudelft.nl/~ruben/thesis_RMSmelik_color.pdf>.

The Pokémon Company. *Pokémon Conquest*. Video game. Nintendo, 2012.

Weiner, Jonah. “The Brilliance of *Dwarf Fortress*.” *Nytimes.com*. New York Times, 21 July 2011. Web. 2012. <http://www.nytimes.com/2011/07/24/magazine/the-brilliance-of-dwarf-fortress.html>.