Divide and Conquer in Genetic Algorithms: Generating Paths on Heightfields

RYAN MYERS
School of Engineering
University of Portland
Portland, Or 97203
U.S.A.

BART RYLANDER
School of Engineering
University of Portland
Portland, Or 97203
U.S.A.

Abstract: - We demonstrate a genetic algorithm (GA) to evolve an efficient path between points in a 3D landscape for a personal computer (PC) strategy game. The GA produces paths preferable to the current state of the practice in game design (most games use a variant of memory-constrained A*). We then compare the fitness of a large genotype that optimizes the complete path between points in the landscape to a strategy of a small genotype that uses the GA to generate a sequence of subpaths between the two endpoints. Surprisingly, the *divide-and-conquer* method statistically produces better paths than the larger genotype. Unfortunately, this is offset by an increased execution time that is considerably larger for the divide-and-conquer approach.

Key-Words: - Genetic Algorithm, Path Algorithms, Game Search

1 Introduction

A common structure in many PC game environments is the heightfield, a finite two-dimensional (2D) array of heights, forming a uniform sampling of a three-dimensional (3D) landscape. Often, games and other applications of heightfields involve moving from one point to another in the most efficient way possible. To reproduce the 2D surface in 3D, a programmer needs only interpolate between the sampling points to get vertices of the surface, and recalculate normal vectors. Since this method is straightforward to program and consumes little space, it's the most common method of storing both real-life landscapes (GSPS elevation maps) and virtual environments. In most cases, heightfields are stored on disk as a 2D full color image, where the luminance of each pixel (scaled to the range 0.0 – 1.0 inclusive) represents the elevation of the landscape.

A common task in many applications of heightfields is plotting a path for an object to move through - for example, a helicopter along a mountain valley, or moving a tank in a war game. However, traditional pathfinding techniques don't lend themselves well to heightfields – the current state of the practice (SOTP) for PC war strategy games is Ensemble Studios' Age of Empires 2, which uses a form of memory-limited A* with very minimal use of the height element of the board. This means that movement tends to move in the most direct way, rather than the most "natural". For example, if you direct a unit to cross a ravine, the memory-limited A* method will cause this unit to rappel down the wall and climb up the other side, even if a bridge crosses the ravine just 50 yards to the north.

A *procedural* approach to the task of path calculation is infeasible due to the enormous search space. Despite this, it's fairly straightforward to quantify what makes a path desirable. Given these characteristics, it seemed that applying evolutionary programming techniques, and genetic algorithms in particular, might be helpful.

2 Fitness Function: Quantifying Goals

In order to create a divide-and-conquer technique within the GA, we need to ensure an even distribution of waypoints along the path. This is fairly trivial to implement. Figure 1 presents a depiction of a high quality path for this goal.

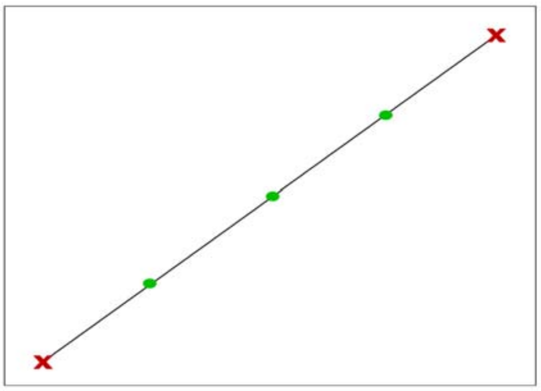

Fig. 1, High quality path.

In contrast, a path with waypoints poorly distributed would not present a suitable representation of a divide-and-conquer technique. Figure 2 depicts a poor quality path.

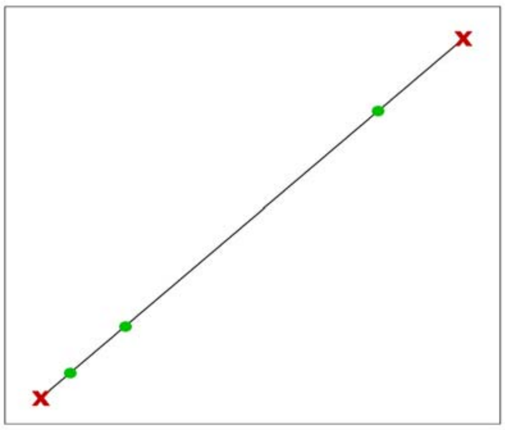

Fig.2, Poor quality path.

Even distribution of waypoints is not the only criteria for a good path. We must also select a path that avoids obstructions. For example, if we know that an enemy city is between the beginning and end points of the path that our unit must take, the most desirable path is one that avoids the enemy by curving around it. See figure 3.

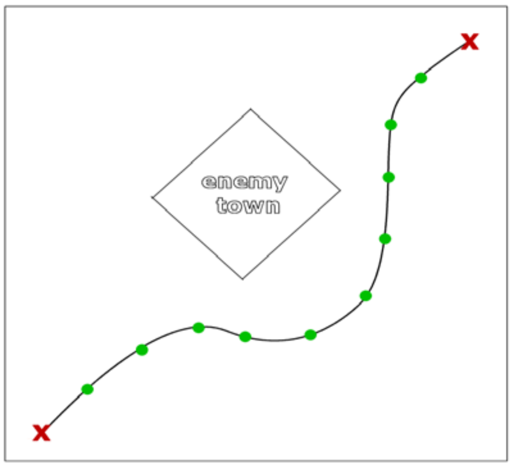

Fig. 3, A path that avoids the enemy.

The other major objective is to minimize unnecessary movement. The first part is to minimize 2D movement. See figures 4 and 5.

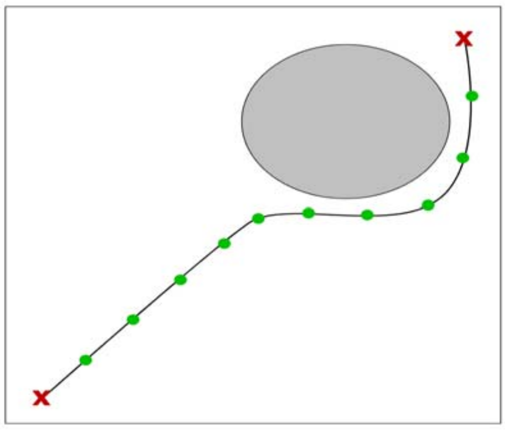

Fig. 4, Preferable solution minimizing 2D movement.

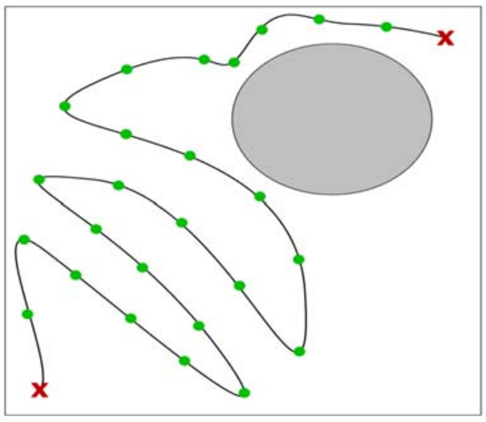

Fig. 5, Poor solution with wasteful 2D movement.

A more important priority in this objective, however, is to minimize elevation changes. Paths that look excellent from a 2D perspective may in fact be very poor choices when considering the players (usually simulated humans) that actually must take the path. Dramatic changes in elevation may offset any advantage in saved distance because of the increase in energy expenditure. See figures 6 and 7 for depictions.

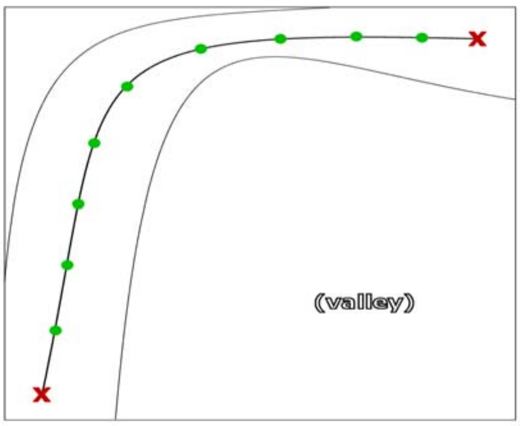

Fig. 6, A good path that *avoids* a valley.

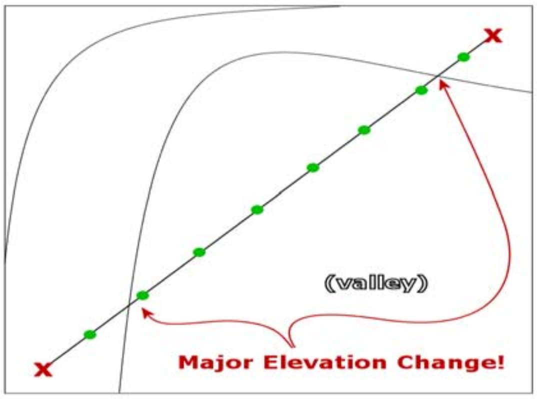

Fig. 7, A poor path that ignores the valley.

Because of the way fitness functions combine, we can write each one of these objectives as an independent function and combine them as a weighted average later.

3 Implementation

The phenotype is a series of unsigned integers split into (x,y) pairs on the 2D heightfield. The decoder simply moves from left to right evaluating 8-bit pieces of the bit string as normal binary integers. This maintains heredity, and does fairly well on locality, although you can still create large changes on a per-node basis if you flip the bits.

Initial testing was done with a population of 500 chromosomes. One percent mutation was used. The crossover function is a fairly straightforward random splice of the parents' genotypes. The parent selection was written to favor the smallest fitness found – the fitness functions measure deviance from goals, rather than grading them on a set scale. In this case, having no upper bound for the fitness did not affect the algorithm's performance. A weighted average was achieved by balancing it against the average of the last average fitness and the last maximum fitness.

While building the algorithm, another issue that came into play was the idea of divide-and-conquer – in PC gaming. Memory-constrained A* is typically used to generate a small number of waypoints (eight seems to be used most often) and then ran a second time to generate two sub-waypoints between each waypoint. In view of this, we also extended the software to include a mode in which the GA was used multiple times – first to generate four or five waypoints, and then to generate four sub-waypoints for each link in the path. Although total operating time is longer than for a single *large* genotype, if each segment is subdivided only when the unit/character is walking in *game-time*, it would be more efficient than generating the entire path in one pass with the large genotype.

4 Conclusions

In general, both the large genotype GA and the divide-and-conquer GA perform very well, especially when compared to the current state of the practice. For the final testing, a freeware fractal-based terrain generator was used to generate 1000 random landscapes, and each landscape was used as input to a population of 2048 64-bit chromosomes. The population was evolved until the difference between the population's average fitness and minimum was smaller than a given epsilon (we used 0.001). On the 1000 random landscapes, the mean convergence is 49.8 generations, with a median of 62 generations.

To provide an example, the terrain below was randomly generated, no enemy groups were placed upon it, and the algorithm was asked to plot an optimal course from one corner to the other (from (16, 16) to (240, 240)). See figures 8 and 9.

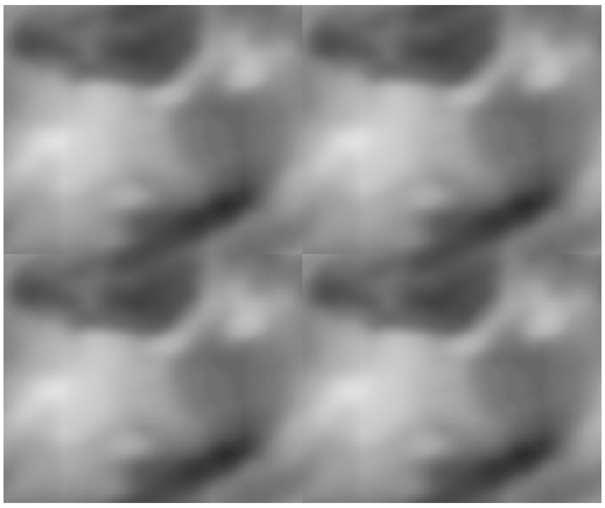

Fig. 8, Random generation of terrain.

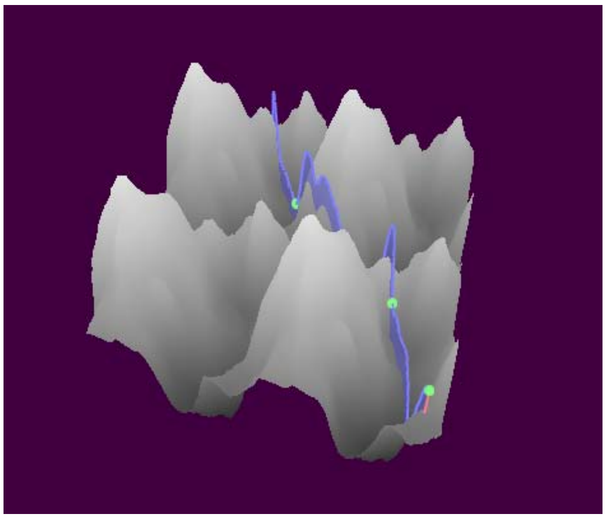

Fig. 9, Rendered terrain with selected path.

As the screenshot demonstrates, the algorithm's path automatically finds the lowest points to cross ridges and peaks in the landscape, and skirts around the two rightmost hills.

Regarding the divide and conquer approach, contrary to intuition, divide-and-conquer produces better paths than using a single large genotype. On average, using the 1000 generated landscapes, the fitness of a 16-point divide-and-conquer path (produced by generating four points with a 64-bit genotype and then generating four more points for each link) was 78% of the fitness of a 16-point path for the same landscape generated by a single 256-bit genotype. (Remember that in this experiment, lower fitnesses represent better solutions.) However, despite this advantage the visual difference is comparatively little, and the execution time for the divide-and-conquer approach is considerably larger.

References:

[1] Rylander, B., Foster, J., GA-hard Problems, *Proc. On Genetic and Evolutionary Computation Conference*, 2000.

[2] Holland, J., *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI, University of Michigan Press, 1975.

[3] S. Russell and P. Norvig. Artificial Intelligence, A Modern Approach. *Prentice Hall, 1995.*