Jesse Chen

Professor Mark Riedl

CS 4731

2 May 2016

Final Report

Planet Terrain Generation

Terrain generation could be accomplished through either a multi-agent approach or the diamond-square algorithm. A multi-agent approach involves using a number of specialized agents operating in parallel to generate terrain: coastline, beach, mountain, river, and smoothing agents. Each agent performs a specific, self-explanatory role in terrain generation. The diamond-square algorithm is a much simpler algorithm that creates terrain by simply generating a height map.

The end result of both solutions is an adequately convincing landmass that most players would not think twice about. However, the problem with the diamond-square algorithm is its simplicity in that it fails to generate complex terrains. While the height maps created can make mountains and valleys with water below a certain elevation, it cannot effectively make things like rivers and beaches. In contrast, a multi-agent approach is capable of generating far more complex and realistic terrain. With each agent responsible for a specific feature, much more detail is evident in the final result because the agents are able to employ more convincingly realistic rules on terrain generation.

While the diamond-square algorithm is very simple to implement, the algorithms necessary for each agent in a multi-agent approach are not much more complex in comparison. The diamond-square algorithm is also faster in runtime but given the expected computing power of the target platform, a multi-agent approach would not lag very far behind. Furthermore, and more importantly, a multi-agent approach would provide players with a more interesting and engaging variety of planets and environments to explore than the diamond-square algorithm would be capable of. As a result, it is recommended that a multi-agent approach be used for planet terrain generation.

Flora Generation

Two possible solutions to flora generation include the use of genetic algorithms and L-systems. Genetic algorithms are a type of search-based generation that mimics biological evolution by iteratively employing random mutations and crossovers in a population to eventually find an optimal individual. L-systems are a type of grammar-based generation that allows for parallel rewriting.

Since L-systems are recursive by nature, they are therefore significantly better suited to making fractal-like figures, such as plants, than iterative algorithms like genetic algorithms. As a result, L-systems are very effective in representing trees in a branching structure similar to trees in real life. Without a very good heuristic to evaluate how tree-like an individual is, genetic algorithms will have significant difficulty generating realistic trees because they are more focused on finding singular optimal solutions than branching like a tree actually would. Additionally, a genetic algorithm would take significantly longer to run depending on the number of iterations/generations desired, among other parameters, thus making it infeasible and impractical for real-time flora generation when L-systems do a better job in a shorter time. Therefore, L-systems would be the best solution to flora generation.

Fauna Generation

Fauna generation can be achieved through a number of different search-based optimization algorithms. Two such examples are simple hill climbing and genetic algorithms. Simple hill climbing is a type of iterative local search that begins with a random possible solution before making gradual modifications to find a more optimal solution. Genetic algorithms, as explained previously, attempt to mimic biological evolution by iteratively introducing random mutations and crossovers in a population in order to search for an optimal solution.

Although simple hill climbing and genetic algorithms are both search-based optimization techniques, genetic algorithms are capable of finding an optimal result much more quickly than simple hill climbing can. This is because simple hill climbing is bound to incremental changes to current samples while genetic algorithms generate new samples completely independently of previous samples. As a result, genetic algorithms can modify multiple characteristics of a sample simultaneously, while simple hill climbing algorithms can only modify one characteristic at a time. Furthermore, simple hill climbing algorithms carry the possibility of getting stuck at local maxima rather than finding the global maximum. As a result, fauna on different planets could end up nearly identical if similar environmental factors exist between those planets. Because genetic algorithms use random mutations and crossovers in a population of samples, they will likely result in a much greater variety in fauna across planets. In a game like *Y’all’ses Sky*, where runtime and visual diversity are critical to the player experience, it would be best to use genetic algorithms in fauna generation.

Alien Spaceship Designs

Alien spaceship designs are yet another game element that could be implemented using genetic algorithms. However, the difference between genetic algorithms for generating alien spaceship designs and generating fauna lie in their respective evaluation functions. Evaluating a spaceship’s design could depend on a number of arbitrary criteria, whereas evaluating fauna generation would generally depend solely on their ability to survive in a given environment. Those evaluation criteria could either be predetermined and universal across all alien races, or they could depend on the alien race, which would also be procedurally generated, that it belongs to. Examples of possible evaluation criteria include symmetry, size, geometry, and so on.

An alternative to genetic algorithms would be to use a Mad Libs technique. Instead of using words to fill in blanks, as in the original word game this technique is based on, premade spaceship modules – such as engines, wings, hulls, etc. – could be used which can then be mixed and matched according to some specified guidelines to generate spaceships. The advantage to this is that developers will have more control over what the resulting spaceships will end up looking like. The downside is that this still requires a significant amount of work to create all of the modules needed to provide a sufficient number of possible spaceship combinations, and players will eventually begin to recognize patterns in the spaceships’ designs across different alien races. The amount of upfront effort required makes it infeasible with such a small team of developers. In the case of a Mad Libs approach to alien spaceship design, the disadvantages heavily outweigh the advantages, and a genetic algorithm should be used instead.

Alien Appearances and Languages

The generation of alien races’ appearances should be more or less similar to that of fauna generation, with the exception of additional constraints in order for them to qualify as descendants of the Precursors. As a result, and by the same reasoning as in fauna generation, genetic algorithms would also be the best solution for generating alien races’ appearances.

Fauna and Alien Pathfinding when on Planets

Hierarchical A\* and the Floyd-Warshall algorithm are two very effective pathfinding algorithms that could be used for both fauna and alien movement on planets. Hierarchical A\* is a variant of A\* that first builds a smaller search space by generalizing a large search space into broad areas rather than using precise locations. A\* is then performed on the smaller search space to arrive at the general location of the target destination, where A\* is performed again for that specific area alone. This significantly reduces the computation time of running a traditional A\* search through, say, an entire planet. The only potential inefficiency with using hierarchical A\* exists if NPCs will be navigating in strictly static environments because A\* is best suited for dynamic environments. Regardless, it is not a significant concern.

In contrast, the Floyd-Warshall algorithm is specialized for static environments in that it precomputes the optimal path between any two possible locations. As a result, there is no runtime computation that needs to be done because it is all done at design-time. However, the downside to this is that there is a very high memory requirement to achieve this. Especially considering that there are a nearly unlimited number of planets, this solution will quickly become infeasible. Furthermore, the Floyd-Warshall algorithm will only work under the assumption that all NPCs will be navigating in strictly static environments. As a result of the limitations of the Floyd-Warshall algorithm, hierarchical A\* would be a better solution for fauna and alien pathfinding on planets.

Alien Tactical Combat Decision-Making When on Planets and in Space

A number of different decision-making techniques can be used to implement alien tactical combat decision-making. Two such effective techniques are finite state machines and task network planners. Both of these methods require designers to develop a set of predefined behaviors. Both can also be used in a hierarchical manner similar to that of hierarchical A\* in that similar and sequential behaviors can be encapsulated into broader, more generalized behaviors. However, although they become easier to think about, finite state machines do not actually benefit from hierarchical encapsulation. A finite state machine gives an agent, an alien or alien race in this case, a finite number of states in which it can be in at any given time – such as idle, chase player, attack player, retreat, etc. Depending on transition rules between states and current facts about itself and its surrounding environment, the agent will decide which state it should be in.

In contrast to hierarchical finite state machines, hierarchical task network planning does benefit from hierarchical encapsulation. They are domain-independent planners such that there can be many different combinations of tasks and ways to complete them that the agent must decide from. Those tasks are decomposed into subtasks which can be treated as goals for the agent to achieve before moving onto the next subtask. For example, the task of hunting down a player could be decomposed into the subtasks of locating, chasing, and shooting the player. The primary advantage that this technique has over finite state machines is that it is capable of re-planning if a currently executing plan breaks. This is done by going back up one level in the hierarchy and re-decomposing. Additionally, hierarchical task network planning can be faster than finite state machines at runtime without much more upfront work, and they can allow for the same alien races to form patterns that the players can recognize and identify with that particular race. Therefore, it is recommended that hierarchical task network planning be used over finite state machines for alien tactical combat decision-making.

Fauna Behavioral Decision-Making

Behavior trees and rule systems are two possible solutions to handling fauna behavioral decision-making. Both techniques are goal-oriented types of reactive planning in which real-time decisions are made by performing an action at every instant. Behavior trees, however, are fast and much simpler to implement which would be a significant advantage given the size of the development team. As its name suggest, a behavior tree is a tree of behaviors that specify what an agent should do under any particular circumstance. While agents operating under a behavior tree appear to exhibit goal-driven multi-step behavior, they are not really capable of responding to unique situations. This is due to the fact that behavior trees are only as good as their designers make them. In other words, agents can only choose to behave from a predefined set of behaviors at any given time that exactly matches current facts about themselves and the world around them.

Rule systems, on the other hand, are difficult to implement effectively and even more difficult to debug. A rule system is a collection of if/then constructs that describe a behavior in terms of facts about the world and the rules of the system. Using pattern matching, the facts are matched to rules, which then determine the resulting behavior that may then change the current facts of the world. Because rules do not need to be matched exactly, this approach is very flexible and capable of responding to novel conditions. Furthermore, less work has to be done in implementing all possible contingencies; however, this depends heavily on the designers’ ability to create a robust rule system, which can be a very difficult task in itself. Considering the scope of *Y’all’ses Sky*, it would be worth the effort in designing a strong rule system that can be tuned accordingly to fit any fauna and handle any given situation. The sheer number of different behaviors that would need to be implemented in all behavior trees of equivalent scope would be infeasible to attain.

Mission Generation

Similar to generating alien spaceship design, mission generation can also be accomplished with a Mad Libs technique. It would be very easy to generate missions that follow a generic template for each type of contract job that the player can accept. For example, “Kill [X] [enemy] on [planet]”, “Search [location] for [Precursor artifact]”, and “Escort [NPC] to [city] on [planet]” can easily be completed by randomly selecting enemies, NPCs, planets, etc. to generate a nearly unlimited number of missions. The problem with using a Mad Libs technique is that such generated missions become stale, boring, and repetitive for players very quickly.

An alternative would be to use a system such as the Stanford Research Institute Problem Solver to give NPCs simple story planning systems in order to make procedurally generated missions more meaningful and engaging for the player. An NPC’s needs or wants can be derived from his or her personality, which can then be used to plan out unique and dynamic storylines in the form of procedurally generated missions that the player can be tasked to complete. This allows for both independent missions and missions that build off of each other. Simple one-off NPC tasks become independent missions, and NPCs with more complex backgrounds will generate a sequence of missions that can change dynamically depending on players’ actions. This approach may require upfront planning for NPCs’ backgrounds, but those can be procedurally generated as well. Due to the simply inadequate nature of using a Mad Libs technique for mission generation, an NPC story planning technique is by far the superior alternative.

Works Cited

Boxley, Paul. "Terrain Generation with the Diamond Square Algorithm." *Paul Boxley*. N.p., 26 Mar. 2011. Web. 28 Apr. 2016.

"Diamond-square Algorithm." *Wikipedia*. Wikimedia Foundation, 18 Jan. 2016. Web. 28 Apr. 2016.

Mitchell, Melanie, John H. Holland, and Stephanie Forrest. "When Will a Genetic Algorithm Outperform Hill Climbing?" Proc. of Neural Information Processing Systems 1993, Denver, Colorado, United States. Neural Information Processing Systems Foundation, Inc. Web. 29 Apr. 2016.