Proposal For *Y’ all’ses Sky* Game Design

*Introduction*

*Y’ all’ses Sky* is a game about space exploration with an approximately unlimited number of planets, which will have their own, unique terrain, flora, and fauna. The player has an interstellar spaceship and makes a living as a freelance-for-hire. Non-player characters (NPCs) will post contract jobs that the player can take.

Since there are so many AI components and our company only has about a dozen employees, we will use procedural modeling and artificial intelligence (AI) to create the game content. There are six main groups of methods for Procedural Content Generation (PCG) that we will discuss and evaluate later, which are Pseudo-Random Number of Generators (PRNG), Generative Grammars (GG), Image Filtering (IF), Spatial Algorithms (SA), Modeling and Simulation of Complex System (CS), and Artificial Intelligence. Due to space limitation, the lists of PCG methods and details are summarized in Appendix A.

*1. Planet Terrain Generation*

The planet terrain should have distinctive continents and oceans. When humans colonize planets or when planets have alien civilizations, they will have cities. Alien civilizations should have distinct architectural and city planning styles. Human cities should look similar to each other. Therefore, the main game content for this part should include oceans or rivers, and city generation, such as road networks and buildings. But the two parts should be generated respectively with different techniques.

The city generation can be made by L-systems (GG) and agent-based simulations (CS), with evaluation from realism, variation, efficiency, and real-time. As a key aspect of city character and identity, road networks adopt some patterns based on the effects of geography, location, cultural influence, etc., while city buildings display a diverse range in both function and style. L-systems, as a grammar-based approach, deploy rewriting for defining complicated objects by successively replacing parts of a simple initial object (Kelly & McCabe, 2006). In the process, a string will be generated by the grammar to describe the characteristics of an object. In order to build road networks, we can control the parameters such as population density, geometric patterns, elevation, and local constraints. In order to construct buildings, we identify the polygonal regions by enclosed streets using roads of the network as dividing borders, and then apply a parametric L-system to refine the building and generate textured faces (See Figure 1 and Figure 2). Several building styles include: skyscrapers, commercial, and residential. A great example for L-system application is CityEngine (Parish & Müller, 2001), which including road generation, building construction and building face creation.

Overall, L-systems are suitable for city generation due to their concise nature, computational efficiency and data amplification properties. They can efficiently generate realistic cityscapes with a good range of road networks, and different building shapes and types.

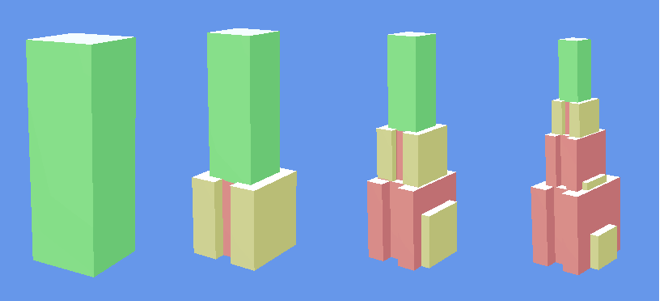


Figure 1: L-System building refinement from bounding box

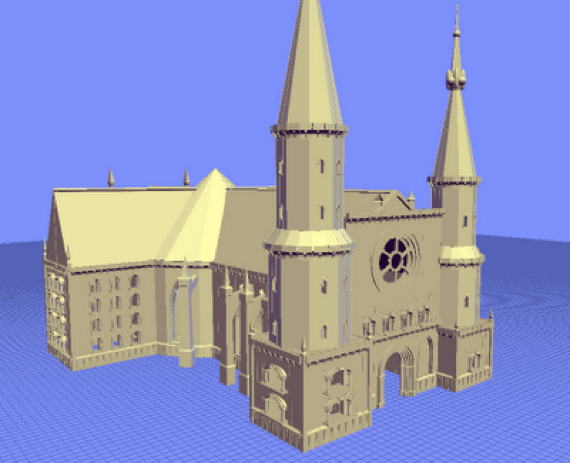
 

Figure 2: Polygonal output of L-system Figure 3: IFS Fractal Ferns

The agent-based simulation is based on modeling a complex situation using agents such as developers, road builders and planning authorities, in which emergent behavior will arise out of relatively simple agent interactions. Roads are built from road segments that are created by two types of agents-extenders and connectors (Smelik, De Kraker, Groenewegen, Tutenel, & Bidarra, 2009). The extender searches for unconnected areas that are not too far from the existing network to find the most matching path and connect the area to network. The connector checks the length of the shortest path from a certain location on the existing network to a randomly chosen spot within a certain radius. If the distance is too long, the connector will propose a direct road connection between the two points. And several developer agents complete the generation of buildings. The buildings are divided into types including not only industrial, residential and commercial, but also special areas like squares, government buildings and institutions. All developers explore to increase the value of their land by evaluating with different rules. For example, commercial developers seek the busiest sections of road networks, while residential developers look for less busy areas. A famous application for agent-based technique is CityBuilder (Lechner, Watson, Wilensky, & Felsen, 2003), which models not only the road networks and buildings but also the growth and development of the city over time. The agent-based simulation has good variability and visualization, but it is time consuming and need high computational complexity.

Compare to city generation, PCG of bodies of water, such as oceans and rivers, is often completed during the generation of outdoor height map by a flooding algorithm. It sets the water level to a specific height and all height below this threshold are considered as under water. SA (Pmsinkiewicz, Prusinkiewicz, & Hammel, 1993) and CS (David, 2004) techniques can be used to model realistic water flow and erosion.

*2. Flora Generation*

Flora is often self-similar. Thus, we can use GG and SA to procedurally model the vegetation. The two techniques I recommend are Fractals and L-systems.

First of all, Fractals, one of the spatial algorithms, include recursive figures that make seemingly infinite detail to be stored as simple recursive functions. We can control a few parameters to get a wide range of possible results. Since natural flora shapes are not easily described by conventional geometric methods, such as ferns are not cones and mushrooms are not spheres, we can describe these shapes using a number of fractal mathematics. One main character of fractals method is the shapes contain a large degree of self-similarity. Thus fractal-like shapes such as ferns or trees can be generated recursively. Instead of using recursive process, which will cause intensive resource, we can use Iterated Function Systems (IFS) to generate vegetation faster using an iterative process (see Figure 3).

Fractal algorithms can change the natural objects with structural complexity to abstraction with various levels of details. There is no theoretical limit to the amount of recursion, so we can create infinite levels of details within each objects. However, due to the limitation of requirement for self-similarity, this algorithm is not flexible and may not be suitable for all kinds of vegetation.

Next, L-systems as introduced above, can be used to define, model and visualize sophisticated plant structures based on grammar rewriting. There are many pre-generated plants types such as trees, flowers, bushes and vines. In addition, planets have certain characteristics concerning climate, temperature, humidity, etc. Each of these properties will be set by upper and lower bounds and will be generated by a random selector. The stochastic process guarantees that the potential plant will be valid and unique. Compare to Fractals algorithm, L-systems is more flexible, but it also has weakness on space/memory consuming.

*3. Fauna Generation*

I recommend simulated annealing algorithm and genetic algorithm for fauna generation, and the former is more preferred.

Simulated annealing algorithm models the physical process of heating a material and decreased the temperature to minimize the system energy, while genetic algorithm uses imitation of biological evolution to get optimization. In genetic algorithm, possible solutions are transferred to strings or chromosomes, and then the quality of a solution can be evaluated by a fitness function. A mutation function converts a solution to a new one, and a crossover function specifies exchange of string parts between numbers of parent strings. The process continues until a sufficient solution is found or until reaching predefined rounds.

The genetic algorithm requires more processing storage and computes numerous instances for a large population set, while the simulated annealing algorithm works on an individual instance at a time. Based on the specification of the typical system running for this game, the genetic algorithm will satisfy the processing limits. Besides, since the genetic algorithm is closer to the actual process of animal evulsion and has more control over the algorithm’s performance through evaluation functions, it will produce better results. Each animal has specific characteristic concerning size, lifestyle, appearance, etc. and these properties can be used as parameters for an evaluation function to specify the fitness of particular generations produced by the algorithm.

*4. Alien Spaceship Designs*

There are several ways to approach this problem each with their pros and cons, and the techniques can be used are Perlin Noise (PNG), Shape Grammars (GG) and Genetic Algorithm.

First of all, perlin noise generates maps of random values by a seeded pseudo random function through interpolation. We can use scaling or combine multiple layers to add more details into the data points. The process includes the generation of several prototype spaceship components, including wings, gears, fuselage, engines, weapon systems, etc., based on spaceship’s functions. Designing the prototypes of these components is similar to creating city buildings but with more details.

Perlin noise can efficiently generate geometry and textures with minimal storage requirements and a few simple parameters. However, this algorithm is suitable for less sophisticated objects and volumetric textures such as mountains and marble vase, but will not generate ideal models for multi-layer and multi-function objects such as spaceship. We can combine perlin noise with other GG and CS techniques (Li, Hoi, Zhao, & Gopalkrishnan, 2011).

Next, shape grammars have its unique rewriting process different from L-system, as it starts with a basic shape and deform and modify it until it resembles the desired model by various operations such as translation, rotation and separation. Thus shape grammar can generate more complex structures. For example, we can define an initial state with three basic symbols of central spaceship-body, wing and top, and then define a number of production rules stored as a derivation tree to generate an actual modeled spacecraft.

At last, genetic algorithm uses mutation and crossover to manipulate the components of spaceship and run them through an evaluation function. The parallelism property allows genetic algorithm to evaluate implicitly many schema at once and particularly suitable for problems with huge potential solutions. Genetic algorithm can also manipulate many parameters simultaneously without previously knowing domain knowledge. However, the question about how to write the good fitness functions and then choose parameters such as the size of the population and the rate of mutation and crossover should be carefully considered. Since shape grammar and genetic algorithm are both inherently parallel in nature, they can be combined together for alien spaceship designs.

*5. Alien Appearances and Languages*

The characteristics of alien appearances are not only based on the characteristics of the planet, but also influenced by the characteristics of other alien races.

For alien appearance, we can generate believable aliens by combining the alien race properties and planet properties. And these patterns are similar to those in fauna generation, so the initial populations of aliens can be generated based on either the designers’ discretion or pre-existing examples. The alternative AI techniques listed above for fauna generation can also be applied here.

Compared to alien appearance, language generation will be a multistep process. We can construct a set of alphabets with complimentary symbols, each of which will be categorized according to their symbol appearances in such a way that alphabets with similar symbols are linked. Examples of sets of alphabets can be taken from languages that already exist, dead languages, or left to the designers’ imagination. We can use the mutation and crossover from genetic algorithms to generate new alphabets. Crossover should only occur among alphabets with similar categorization to help insure language cohesion.

An alternative for generating alphabets is to compile a single list of all symbols and generate alphabets at run-time by either randomly selecting symbols or by selecting symbols that share some correspondence. The problem with the first approach is that it can generate languages with low cohesion between characters, and the second approach will face costly processing. As these languages are fictitious and not particularly scrutinized by players, the least amount processing needed to generate plausible results is desirable. Next, the designers can also construct a set of grammars for words constructing. Again examples can be taken from pre-existing languages or the designers’ choice. Mutation and crossover will be used to generate new grammars. Thus, new languages can be created at run-time and assigned to each alien race. The ‘universal translator’ will build a mapping of each English word needed for the game to a word generated by the grammar. Then we can easily display a simple process of mapping between the English texts and alien texts to players.

*6. Fauna Behavioral Decision Making*

Fauna decision-making can be done a couple of ways. Two recommendations are state machines and hierarchical planners.

As animals are typically not particularly scrutinized, the logic that controls their behavior does not need to be particularly sophisticated. Therefore, state machines for different types of animals will be suitable. For example, each animal generated will have a set of behavioral and physical traits. So state machines can be constructed to reflect the typical behaviors of a predatory animal or a herding animal. Some of these states may include, seeking food, running from predator, hiding, sleeping, foraging, etc. In order to make the behavior more distinctive and less predictable, we can set some probabilities for realization of each state’s execution. The element of randomness leads occasionally to animals’ suboptimal behaviors. This is ideal because players do not expect animals to always make the optimal choices.

The alternative mentioned is hierarchical planners, and an example is called hierarchical task network (HTN). HTN, as a planning-based system, is very useful when dealing with complex decision-making. However, the planning process requires much more overhead than necessary and takes a lot of computational resource, HTN tends to be very slow, especially for this game with quite large number of animals that can exit on a planet. The long processor time makes this technique more inappropriate than state machines.

*7. Fauna and Alien Path finding*

The AI recommendation for fauna path finding is deeply related with the constraints that were introduced in the fauna behavioral decision-making section. Based on those techniques, we can minimize the processing and storage requirements to generate fauna actions and reasonable behaviors. Therefore, path finding can be achieved by using virtual potential fields based on the current state of the animal. For example, if a creature is currently seeking food, different food sources around the animal’s location will have a certain potential to lure the animal towards it based on how far the distance away from the animal the source is currently located. As the state of the creature changes, the potential fields for each source also change. Areas of negative potentials can also be built so that it will be much more unlikely to have the existence of the animals at those fields. In addition to this simple ray casting as the fauna move with the pull of the potential fields and the behavior of the fauna can appear much more believable. An alternative that is also considered is the random movement behavior by the animals with ray casting to avoid obstacles or anything that is in the way. Because the random movement would be a very lightweight implementation, this alternative will provide us little control over the different behaviors of the generated animals and most likely lead to illogical or strange gameplay environment. Therefore, random movement shouldn’t be implemented here.

On the other hand, alien path finding needs to be more sophisticated, because sometimes aliens can be NPC characters and the players would expect them to be much smarter than the animals. To accommodate those factors, our recommendation is to use A\* on a navigation mesh of the planet along with ray casting to avoid collisions and then smoothing to optimize the path. By creating a nav-mesh of the planet A\* will be able to find the optimal path. The very interesting question at this point will be what will be the heuristic function to produce the best path. This function should account for potential factors including potentially dangerous entities in the environment such as dangerous animals, other alien races, enemy vehicles, and more. Each of these factors should make the heuristic values with different coefficient, and this value will lead to an optimal path under different distance and environment. Another potential method to determine path planning would be to discretize the space and transform the space into a grid. While this method could be more desirable here to be used here as the path planning algorithms become simpler and much easier to implement, the resulting path is often not as smooth and aren’t optimal after all. Because the alien behaves is an integral element and important part of the overall game experience the former techniques are recommended over grid navigation because of the better generation of optimal path.

*8. Alien Tactical Combat (On Planet/In Space)*

For tactical combat behavior of aliens, there are several different AI techniques to choose from. Some of these include partial order planning, hierarchical task networks, or behavior trees. Ultimately, the recommended algorithm is a hierarchical task network because it combines both hard coded domain knowledge and run time planning. In terms of domain knowledge, each alien race has a specific “personality” or racial behavior pattern, which is listed as traits. These traits can take an account into an evaluation heuristic to determine which path a plan will take in the task network. For example, an alien race with high intelligence and low aggression will typically choose to attack less, utilize cover more, and generally behave more strategically. And on the other hand an aggressive race may adopt a strategy of an all in attack or brute force under every situation. By making domain knowledge to influence the planning process, we can allow the races trite to match their behavior in combat. This is more desirable than the method of behavior trees in that behavior trees need much more hardcoded design at design time and behavior trees is hard at handling edge cases under different situation.

*9. Mission Generation*

The recommendation for mission generation is a combination of methods of partial order planning and goal driven NPCs. The advantage of goal driven NPCs is that the missions generated in this way will give players a sense of much less arbitrary and, and instead the player can have a much better game experience in term of the storyline within the environment. An alternative is to use procedural systems that use the “build your own” approach where a quest is selected from one set, a difficulty level is selected from another, and a reward is picked from yet another set. This method will require not only significant more design time resources and game memory to generate enough mission content for the game experience, but also it is susceptible to generating discrete and fragmented missions that struggle to tie a storyline together. Moreover, partial order planning will also be utilized to help in the mission generation because the technique will generate quests using both domain independent heuristics and domain dependent heuristics. As a result, a domain independent heuristic will seek for the fewest number of actions to create the best plan while a domain dependent heuristic will prefer plans with specific actions in their solutions. By adopting this technique, missions can be given out as specific tasks that are required and can be completed with a reasonable number of actions.

Reference

David, C. (2004). GameDev. Retrieved from http://archive.gamedev.net/archive/reference/articles/article2065.html

Kelly, G., & McCabe, H. (2006). A survey of procedural techniques for city generation. *ITB Journal*, 87–130.

Lechner, T., Watson, B., Wilensky, U., & Felsen, M. (2003). Procedural city modeling. *1st Midwestern Graphics Conference*, 1–6. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.8874

Li, B., Hoi, S. C. H., Zhao, P., & Gopalkrishnan, V. (2011). Procedural Content Generation for Games: A Survey. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, *15*(212), 434–442. http://doi.org/10.1145/0000000.0000000

Parish, Y. I. H., & Müller, P. (2001). Procedural Modeling of Cities. *28th Annual Conference on Computer Graphics and Interactive Techniques*, (August), 301–308. http://doi.org/10.1145/383259.383292

Pmsinkiewicz, P., Prusinkiewicz, P., & Hammel, M. (1993). A fractal model of mountains and rivers. *Graphics Interface*, (May), 174–180. Retrieved from http://algorithmicbotany.org/papers/mountains.gi93.html

Smelik, R. M., De Kraker, K. J., Groenewegen, S. A., Tutenel, T., & Bidarra, R. (2009). A survey of procedural methods for terrain modelling. *3AMIGAS - 3D Advanced Media In Gaming And Simulation*, (June 2015), 25–34. http://doi.org/10.1145/1814256.1814258

A. Details Of Methods For Procedural Content Generation

A.1 Pseudorandom Number Generators (PNG)

Perlin Noise, etc.

A.2 Generative Grammars (GG)

Lindenmayer-systems (L-systems), Split Grammars, Wall Grammars, Shape Grammars, etc.

A.3 Modeling and Simulation of Complex Systems (CS)

Cellular Automata, Tensor Fields, Agent-based Simulation, Other Complex Systems and Theories, etc.

A.4 Spatial Algorithms (SA)

Tiling and Layering, Grid Subdivision, Fractals, Voronoi Diagrams, etc.

A.5 Image Filtering (IF)

Binary Morphology, Convolution Filters, etc.

A.6 Artificial Intelligence (AI)

Genetic Algorithms, Simulated Annealing Algorithms, Artificial Neural Networks, Constraint Satisfaction and Planning, etc.