CONTROLLING CHARACTERS USING PROCEDURAL GENERATION

Aman Soni, Satyam Tyagi, Prerna Jain, Vishal Jha, Ms. Deepika
Student, Assistant Professor
Department of Information Technology
Dr. Akhilesh Das Gupta Institute of Technology and Management

Abstract: ***In gaming and media, it takes a lot of time and effort to design and that is why a constant development in content-generating algorithms is made. With the introduction of procedure generation algorithms, it has become easier to create more data but with less significance.***

In this document, we are processing the possibility of the evolution and generating combinations of procedural generation with new algorithms and implementation techniques, which makes it possible to achieve more realistic behavior with a wider spectrum of predicted actions of non-player characters.

The following research is documented based on the conclusions drawn from a Unity3D self-developed simulation puzzle car game (named as Hit No One) which 'uses the data provided by the user in real-time to control nonplayer characters' to demonstrate the concept, conduct the research and make fruitful conclusions.

I. INTRODUCTION

The concept of Procedural generation is not new. Procedural generation algorithms are in use from a long time in the field of media and game designing where it is being used to create a large mass of data with relatively low significance. In computing, procedural generation is a method to generate data or information by the usage of algorithms and which usually infuses manually provided assets with algorithmically generated assets to produce a new amount of data that may or may not resemble its ingredients. In games, this concept is used to generate a large amount of data that otherwise would have taken much more time and effort to create manually. E.g. - terrains, ground, grass, water etc. are designed using such algorithms.

Procedural generation works on the concept of randomness created by algorithms which can be amplified or reduced depending upon the implementation. Though this is used to create large amounts of data with very little difference between entities, the same can be used to generate data for bigger and more complex entities which may or may not use the same data.

II. CONCEPT

Procedural generation for non-player characters is different from its general form as there is a lesser amount of data generated for more significant entities. Therefore, the data to be generated depending upon the relations, similarities, and differences in their functionality and design. This somehow narrows the application of the same and is the very reason procedural generation is not implemented on many important characters.

But, the implementation of procedural generation is highly dependent on the algorithm designed and thus such algorithms can be developed to create more distinctive data for different entities. Though in some cases where the entities have very similar structures and functionalities, procedural generation can be implemented easily as the algorithms would not have to produce many distinct data.

III. DESIGNING

The designing of algorithms for generating characters and their functional properties is the topmost priority which is followed by the designing of the surrounding environment consisting of levels and other objects.

For this particular case where the characters have similar characteristic properties, a singular algorithm can be applied which can be implemented as a container containing various properties in form of functions, variables or procedures which can variate or mutate depending upon their inheritance in different objects. It can be understood as providing raw data to the algorithm that generated different information sets after considering different characters. Since the generation in real-time, the data so generated should be readily converted into instructions and given to those components who need it and this should happen efficiently in the least frames of time. Also, the algorithm is supposed to generate the characters before giving any instructions and should also either create or check for all properties that are required for a character to function.

IV. IMPLEMENTATION

The above-explained blueprint of the algorithm can be best implemented as a finite state machine that computes a set of instructions. There are several ways to implement the same and we have used some of them.

1. Prefabs, which are fully configured game object components, are fabricated to create characters (cars in this case). Multiple prefabs were designed in order to find the right quality and configuration of characters.
2. The instances of enemy players are created as soon as an instance of a real player character comes into existence. All the other instances try

to detect the instantiated entity (by using tags in this case).

3. As soon as the real-time player character spawns in the playing environment, the rest of the entities will receive properties based on the current properties of the real-time player (in this case the non-player characters are copying player properties such as acceleration, brake torque, and steering, etc.) and will pass these values for their own set of instructions.
4. This was the manual input given by the user. The algorithm consists of functions that will keep generating random values which are given to other functions that utilize both manual and algorithmic data to generate values for various properties (like position, torque, etc.). Here procedural generation has been applied as the values are generated partially by both manual and algorithmic data which is also affected by some predetermined factors of the algorithm.

V. APPLICATION

Though procedural content generation is being widely used these days to produce content for games and graphical media, this particular deviation of procedural generation can create content in real-time instantly. This property can help in generating data that does not need to look for previously existing data to perceive accuracy in behavior or functioning. Therefore such algorithms can be efficient without retraining the model on some data though retraining can be used to increase the efficiency of the algorithm by pressing algorithmic calculations.

This form of procedural generation can be used to control simpler entities that have a pseudo-infinite number of actions possible as they have only a fixed number of available actions (like moving etc).

VI. FUTURE PROSPECTS

Real-time procedural content generation can be efficiently used in situations where a pseudo infinite state is required. That being said, it can also be used in games where the actions of various characters are either based on a particular set of actions or are limited to certain combinations.

Games based on concepts of tower defense, strategy, planning, etc can use real-time procedural generation to generate data based on the real-time actions of the user and carry forward the calculations accordingly. The same can also be used to manage side characters which only have a finite set of actions which can be correlated to other characters movement or actions.

This can also be combined with multiplayer gameplay where the real-time characters input can be utilized to control over a larger amount of non-player characters.

VII. CONCLUSION

The output generated from real-time procedural generation heavily depends upon

the algorithm so developed, the probability of both the probable actions and the real-time actions of the user. Since the set of actions can be unlimited and may or may not be predictable every time, the algorithm used in such cases will have to be advanced and complex in order to generate accurate and precise data for the other characters to be controlled.

Since this particular paper deals with the controlling of characters as well as creating them, there is also a need to define default actions and values and some exception handling which is always important.

Though procedural content generation can be considered a very complex ideology to be utilized in gaming, it is known for creating data in very large amounts with lesser manual inputs and the same can be modified to control the characters without many manual instructions or heavily designed algorithms. This not just decreases the time of development but also saves space and dissolves the complex implementations of algorithms to some extent.

VIII. REFERENCES

[1] Stefan Greuter, Jeremy Parker, Nigel Stewart, Geoff Leach, RMIT University, Melbourne, Victoria, Australia.

[2] Barbara De Kegel and Mads Haahr, IEEE.

[3] Nicolas Marechal, Eric Galin, Adrien Peytavie, N Maréchal, Eric Guérin. Procedural Generation of Roads. Computer Graphics Forum, Wiley, 2010, 2, 29, pp.429-438.ff10.1111/j. 1467-8659.2009.01612.xff. ffhal-01381447