УДК 004.921+004.8

M. I. BEZMENOV, Y. O. POTAPENKO, K. O. DVORNIK

A UNITY 3D ENGINE PLUGIN FOR CREATING STATIC ECOSYSTEM IN GAME APPLICATIONS

Here we report on the researches and development of the static ecosystem plugin to Unity 3D game development platform, also the creation of neural network has been described. It allows for the designer-driven automatic generation of computer game assets based on the two vastly different approaches: procedural and artificial-neural-network-based; with user-defined object to be cloned, area to be populated and placement rules. Both methods have been applied to the problem of photorealistic distribution of stones on the hillside (including demonstration of the common placements mistakes). All the approaches were then evaluated by the panel of the computer gamers. Opinion of some participants of the experiment with the corresponding results were summarized.

Keywords: Unity 3D, game environment, artificial neural networks, procedural generation, computer graphics, 3D modeling, landscape.

Introduction. Game industry exists a lot of years, but its popularity is only growing. One of the most important aspects of the modern game development is creating an in-game environment. Game environment it is something like field, forest or town and all the objects around you in this location (if compare it to the real world). For location looks like real, it is necessary to put environment objects in similar positions, and mount as on the existing location. So, in-game environment consists of a set of a 3D objects placed in certain points of space according to some objective or subjective rules. There are two major approaches to the in-game environment development: procedural and manual [1]. The former is a fully automated process that distributes objects in space according to the well defined rules. The main advantage of this method is practically instantaneous environment generation, e.g. millions of objects could be scattered for less than a second using modern computer hardware. However procedural approach is typically considered to be "unattractive" to the prospective audience of the game due to the relative low variance of the object distributions this methods offers. It is essentially limited by the amount of rules programmers could implement in software on feasible timescales. Consequently, level designers spend considerable amount of time to generate reasonably vibrant in-game environment using trial and error approach on the number of parameters the procedural generator has. The manual approach is obviously even more time consuming since every object (most likely out of thousands) should be placed, rotated and scaled by hand. However as this method fully relies on the artistic impression and skills of the level designer, it might lead to creation of truly unique in-game environments. So the most optimal approach is to use procedural and manual methods together: automatically generate an in-game environment and then refine it manually until it fits the quality criterion of the given computer game project. Here we demonstrate how the artificial neural networks [2] could be used to allow level designers to create their own, unique, set of rules for procedural generators. Such rules then could be trained even further by other designers allowing for vibrant and unique in-game environments to be generated automatically.

The aim of this work is to develop a plugin for Unity 3D engine that uses artificial neural networks to combine the procedural and manual methods of in-game environment creation.

Problem definition. The plugin should allow one to access the landscape system of the Unity 3D engine. So any particular environmental object could be selected to be instanced and distributed in space according to the custom-defined rules with a set of parameters exposed to the level designer. To account for domain-like structure of the real world the plugin should also allow for the distribution area to be user-defined.

The distribution rules set for the given object and (or) its existing placement we would call an ecosystem. For the reference we would rely on the ecosystem created using VUE eON software, which is a state-of-the-art tool widely used in film and CG industries to create and visualize large-scale artificial worlds. We would restrict its procedural generator parameters to the certain ranges of elevation, tilting, rotation and scaling of the objects.

Finally, we empirically chose a particular type of artificial neural network and then estimate the required amount of neurons and synapses. The amount of inputs of the network is kept the same to what we used in Vue eON software, with the input parameters normalized to the respective ranges [2]. The tilt and elevation of the object is estimated from the topology of the landscape using the 8-neighbors method [3]. We employ an open-source NeuralDotNet library that allows one to create, train and apply deep artificial neural networks with backpropagation and dynamically adjustable amount of neurons.

Literature review. Any procedural generation of in-game assets includes natural and artificial objects as emphasized in bestselling Bill Fleming's "3D Photorealism Toolkit". He intentionally splits his book into two parts: creation of city and natural environments; each with vastly different approaches and specific rules that significantly contribute to the realism of the result [4]. Let us consider procedural generation of city landscape. The design of artificial objects typically obeys high degree of symmetry. For instance cities typically expose straight, rectangular and systematically placed objects with minimum of entropy. However, it is not necessarily mean that it is vanishing, e.g. buildings have different heights and there are certain amounts of waste and traffic on the streets. Therefore, procedural generation of city landscapes should combine both chaos-driven and systematic rules. An example of automatically created city environment is shown in Fig. 1.

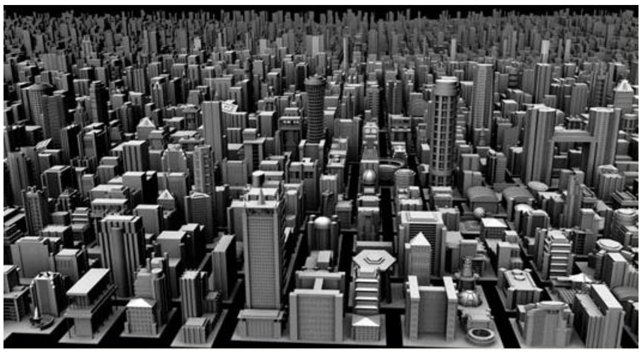

Fig. 1 – A city landscape generated by the procedural approach

To create a photorealistic virtual city with VUE 3D one needs to use a set of different types of buildings. In the VUE ecosystem this could be finely adjusted by pressing the Add Layers, adding objects to populate. In contrast, the rotations should be restricted to multiples of π/2. Such restrictions on the parameters are not supported by the ecosystem, but should be set manually prior to instancing. The random shift of the object should be confined to some small in-plane values as buildings are relatively aligned with respect to each other and, obviously, cannot be elevated above the ground surface. The local surface normal should not be used in procedural generation of virtual cities as buildings are typically made strictly vertical. This set of rules could be extrapolated to any artificial objects, e.g. western gravestones, belts, power grids, etc. A vastly different set of rules should be applied to natural environments. In case the chaos dominates over uniformity. Moreover, the natural object are typically placed along the local normal of the ground surface and there is no need to constrain the rotation. Anyway some empirical placement rules could still be identified. For instance, there is always more vegetation around the large stones, since they are able to keep more moisture; small stones are typically seen in steppes where they are largely affected by the weathering. It is virtually impossible to account for all such rules, but over the years game designers identified the most important of them that lead to the satisfactory level of photorealism. The population of chaotically (without any particular rules applied) placed stones is shown in Fig. 2.

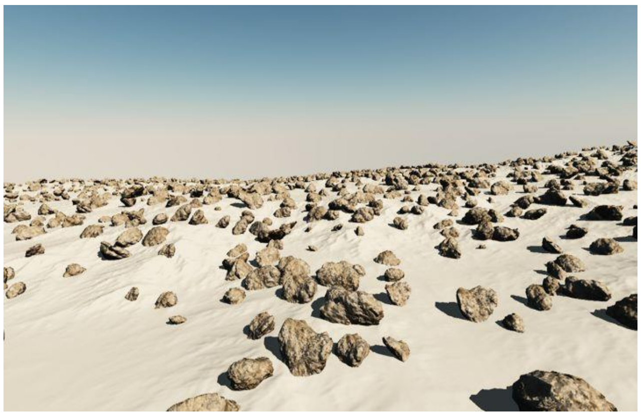

Fig. 2 – A population of stones generated by the procedural approach

Visually the given environment is rather unrealistic. However if a simple rule of proportional to the object size distribution is applied the photorealism of the environment subjectively increases multifold as shown in Fig.3. This particular method relies on golden ratio principle, i.e. the so-called ‘rule of five”. In particular it postulates that the ratio between the nearest neighbor objects in parameter space should be ⅕, e.g. the amount of middle-size stones should be 5 times larger than the large ones. In the reported ecosystem this method is provided by the ‘Make Nice’ function that provides photorealistic distribution of the given objects by the 3-pass instancing, each with corresponding downscaling of the objects and increasing its amount by a factor of 5. The same technique works equally well for vegetation.

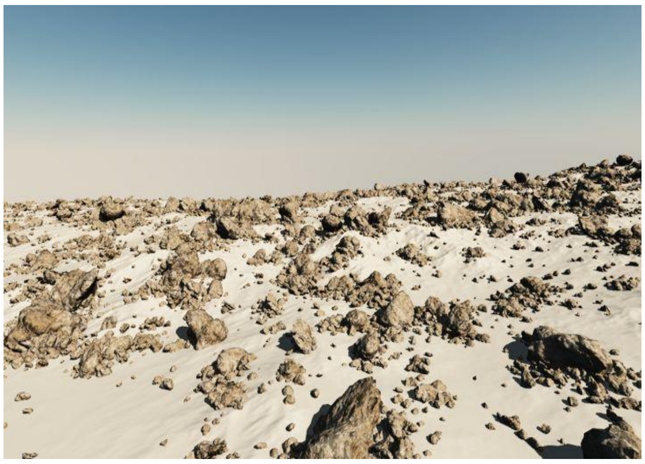

Fig. 3 – Photorealistic distribution of the stones

The proposed solution. In the given work the artificial neural network is used as black box that maps object distribution rules developed by the level designer to the input parameters space. Hereafter we rely on the backpropagation method to train the artificial neural network [5]. A supervisor is then creates a training set with the aim to place the objects in the certain way. Let us assume that one wants to distribute cubes, so that their size increase with the altitude. If the linear dependence is sufficient, then the training set is simply a set of two cubes: one small and one large cube at low and high altitudes, respectively. The input parameters should include altitude of the ground surface at the object site and the corresponding 8 nearest-neighbour heights to account for the rules related to the direction of the local normal to the surface. There should be at least 6 outputs of the artificial neural networks to account for 3D scaling and rotation of the given object. The schematic representation of the described artificial neural network is shown in Fig. 4.

The h_i is the altitude of the neighbor vertex of the ground surface and i = 1, 2..8. (sX, sY, sZ) and (rX, rY, rZ) are the scaling and rotation vectors, respectively. The inputs of the network are normalized independently using the corresponding minimax values across the whole training set.

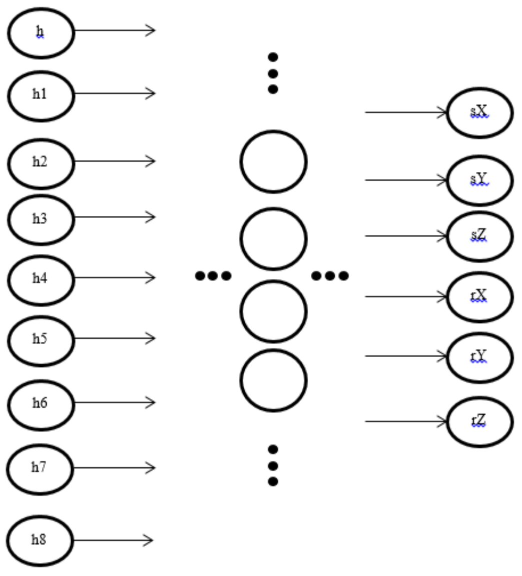

Fig. 4 – Schematic representation of the artificial neural network used to distribute the objects with linear scaling of their size vs altitude

The routine. At first, one needs to put our plugin to the 'Editor' directory of the given game project. Then the plugin should be activated from within the Unity 3D Editor by invoking the 'Terrain->Ecosystem' menu. In the corresponding window the user should specify the object (by sending it to the 'prefab control') and then the area it should be distributed to. Multiple objects could be cloned at once. In addition, the is a possibility to adjust the parameters of the ecosystem: rotation, scale and shift ranges, tilt rules with respect to the global and local normals of the ground surface. Once changes are made the instancing could be performed immediately. To train the artificial neural network, one can chose an arbitrary number of (manually distributed) objects in the scene and proceed. Finally the existing population of the objects could be selected and then re-distributed using different artificial neural network.

Implementation of the artificial neural network. If the given multi-layer artificial neural network relies on linear activation functions, then due to the associativity of the matrix product it could always be reduced to the single layer topology. At the same time if the nonlinear activation function is used then it could be demonstrated that the two-layer network topology is sufficient to construct a universal function approximator [5]. Since we rely on the NeuralDotNet library that only supports nonlinear sigmoidal activation function, then the problem of the particular choice of network topology is, thereby, trivial. So we would rely on the two-layer artificial neural network with backpropagation training routine. At the same time the amount of neurons to be used should be identified using heuristic methods [5]. We would start with 18 (the value is selected randomly) neurons per each hidden layer of the network. To estimate the validity of the this approximation we would calculate the variance of the outputs of the network for the given training set, i.e. the training error:

$$H = \frac{1}{2}\sum_{\tau \in v_{out}} \left(Z(\tau) - Z^*(\tau) \right)^2, \qquad (1)$$

where $Z^*(\tau)$ – is the expected output value of the network, τ – input signal [5]. To avoid the overtraining effect that leads to suppression of the generalization properties of the network the vanishing values of the training error should be avoided. This could be achieved by adjusting the amount of neurons per layer and (or) by restricting the amount of training cycles. At first let us limit this value to 5000 and then investigate the relation between the training error and the amount of neurons per layer.

The training set consists of 50 objects that are randomly scaled and placed along the local normal of the ground surface. By doing so we would make that all inputs and outputs of the network are active. The dependence of the training error on the amount of neurons per layer is shown in Fig. 5.

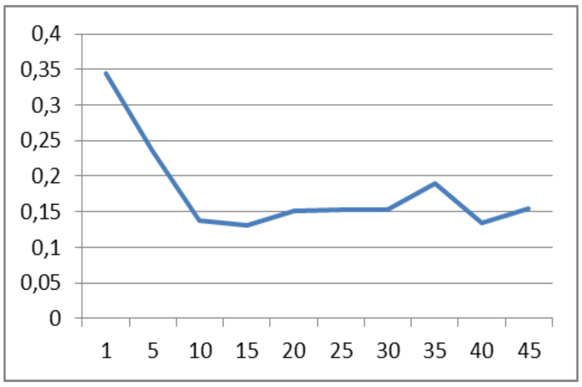

Fig. 5 – Training error vs the amount Ïof neurons per hidden layer

The data clearly shows that the optimal (minimal) value of the training error corresponds to 15 neurons per layer. Beyond this value the training time increases significantly, while the error remains virtually constant. So by fixing the amount of neurons to the optimal value, we investigate the relation between the error and the amount of training cycles. We should also take into account the dependence of the time required to complete the training on the amount of cycles. For instance, 100 and 20000 training cycles require 5 s and 180 s, respectively. The result of this investigation is shown in Fig. 6.

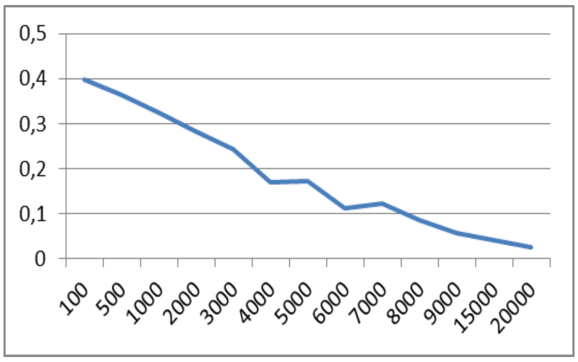

Fig. 6 – The training errors vs the amount of training cycles in case of network of 15 neurons per hidden layer

As it could be seen on the graph, the dependence is linear. So we can easily find the required amount of training cycles for the desired value of the error. The final part of the present study focuses on identifying the subjective criteria of the distribution vibrance produced by the artificial neural network approach as compared to the procedural generation.

For this purpose we create a population of objects that are placed in space using the following equations:

$$SX' = SX * R(o), \tag{2}$$

$$rX = R(o), \quad o \in (-360, 360),$$

where o – the tilt angle. Then we train the artificial neural network accordingly (Fig. 7 – 9). In this example, we wanted to create rocks that fall or are on the slope of a mountain. As the stones were used primitive Box, which is embedded in Unity 3D.

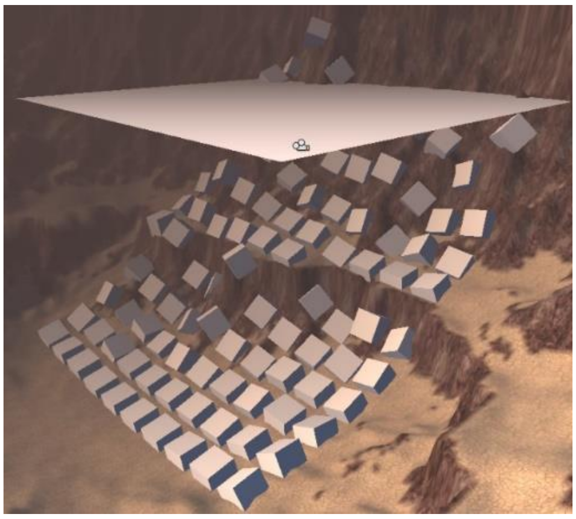
Fig. 7 – Procedurally generated population

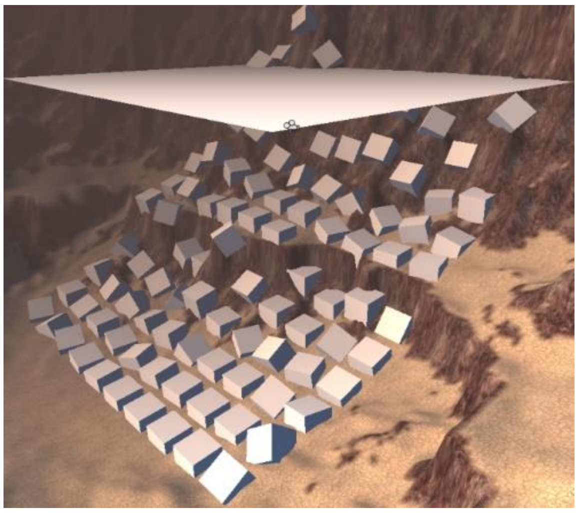
Fig. 8 – Neural network with generation error 0,02486

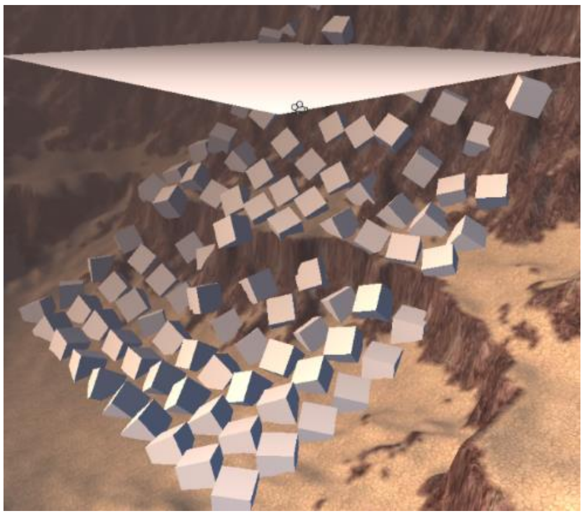
Fig. 9 – Neural network with generation error 0,336326

Fig. 7 shows procedurally generated population, it is clear that not all primitives scattered, the last row are almost not expanded, and has not changed its position. This result does not seem realist for this task [6]. On the Fig. 8 neural network generates better spread "stones" on the slope, but the some objects again, almost did not change theirs angles. Neural network generation with value generation error 0.336326 (Fig. 9) is the best results.

Then we perform a blind experiment (table 1).

Table 1 – The subjective preferences of the experts with respect to the given distribution of the objects

Expert	Generate types of creating game levels		
	Procedural	Neural networks 0,02	Neural networks 0,33
Iliya (Doctrina)			+
Denis (Program Ace)			
Helen (Program Ace)			+
Denis (Doctrina)		+	+
Pavel (CFT)			+
Olexandr (Doctrina)	+		
Mykola (GU, SE)			+
Andriy (BSK Games)			+
Viktor (ideus)			+
Evgeniy (Plaiko)			+
Artur (Plaiko)			+
Roman (Doctrina)			+
Katerina (Doctrina)			+

To assess the obtained results we formed a group of experts consisting of prospective computer gamers and game designers. Experts chose the most visually attractive distribution of the objects (table 1).

Denis Director of Cross platform Development Department at Program-Ace, explained his choice, the generation with error 0.336326 looks like spread and falling rocks. Andrew level designer and CEO of BSK Games, about generation said that from the point of level design view the best is neural networks generation 0,336326, but if the objects would be randomly scattered with bigger value of parameter, would be even better. According to the table we can say that almost all respondents declared by the neural network generation with the value of 0.336326 errors. The procedural and neural network generating have similar results, but the most interesting landscapes created through neural networks generation with the biggest mistake.

Conclusions. We report on the development of an artificial-neural-network-based ecosystem to distribute objects on the virtual, in-game landscapes created in Unity3d editor. In contrast to rather limited procedural generation, our approach allows one to program any custom distribution rules in natural and intuitive ways with fine control over the entropy, e.g. ranging from rather strict to practically random placement of the objects.

Bibliography: 1. *Уоссермен Ф.* Нейрокомпьютерная техника. Теория и практика / *Ф. Уоссермен.* – М. : Мир, 1992. – 184 с. **2.** *Вороновский В. К.* Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / *В. К. Вороновский, К. В. Махотило, С. Н. Петрашев, С. А. Сергеев.* – Харьков : Основа, 1997. – 112 с. **3.** *Заенцев И. В.* Нейронные сети: основные модели. Учебное пособие по курсу «Нейронные сети» для студентов 5 курса магистратуры кафедры электроники физического факультета Воронежского Государственного университета / *И. В. Заенцев.* – Воронеж, 1999. – 76 с. – Режим доступа : http://neuroschool.narod.ru/books/zaencev.html. – Дата обращения : 20 января 2015. **4.** *Флеминг Б.* Фотореализм. Профессиональные приемы работы / *Билл Флеминг.* / М. : ДМК, 2000. – 384 с. **5.** Хайкин С. Нейронные сети: Полный курс / *Саймон Хайкин.* – М. : Вильямс, 2006. – 1103 с. **6.** *Rouse R.* Game Design: theory & practice / *Richard Rouse.* – Plano : Wordware Publishing, 2005. – 698 p.

Bibliography (transliterated): 1. Wasserman, Ph. *Nejrokomp'juternaja tehnika. Teorija i praktika*. Moskow: Mir, 1992. Print. **2.** Voronovskij, V. K., et al. *Geneticheskie algoritmy, iskusstvennye nejronnye seti i problemy virtual'noj real'nosti*. Kharkov: Osnjva, 1997. Print. **3.** Zaencev, I. V. *Nejronnye seti: osnovnye modeli. Uchebnoe posobie po kursu «Nejronnye seti» dlja studentov 5 kursa magistratury kafedry jelektroniki fizicheskogo fakul'teta Voronezhskogo gosudarstvennogo universiteta*. Voronezh, 1999. 76 p. Web. 20 January 2015 <http://neuroschool.narod.ru/books/zaencev.html>. **4.** Fleming, B. *Fotorealizm. Professional'nye prijomy raboty*. Moskow: DMK, 2000. Print. **5.** Haykin, S. *Nejronnye seti: Polnyj kurs*. Moskow: Vil'jams, 2006. Print. **6.** Rouse, Richard. *Game Design: theory & practice*. Plano: Wordware Publishing, 2005. Print.

Received 05.02.2015

Bezmenov Mykola Ivanovych – Candidate of Technical Sciences (Ph. D.), Docent, National Technical University "Kharkiv Polytechnic Institute", Professor at the Department of Systems Analysis and Control; tel.: (057) 707-66-54; e-mail: bezmenov@kharkov.ua.

Безменов Микола Іванович – кандидат технічних наук, доцент, Національний технічний університет «Харківський політехнічний інститут», професор кафедри системного аналізу і управління; тел.: (057) 707-66-54; e-mail: bezmenov@kharkov.ua.

Potapenko Yuliia Olexandrivna –National Technical University "Kharkiv Polytechnic Institute", student. tel: 063-244-64-33; e-mail: yulya.potapenko@gmail.com.

Потапенко Юлія Олександрівна – Національний технічний університет «Харківський політехнічний інститут», студентка; тел.: 063-244-64-33; e-mail: yulya.potapenko@gmail.com.

Dvornik Kostiantyn Olexandrovich – senior game developer, BSK Games; tel.: 093-68-68-195; e-mail: kostiantyn.dvornik@gmail.com.

Дворник Константин Александрович – сеньйор розробник ігор, BSK Games; тел.: 093-68-68-195; e-mail: kostiantyn.dvornik@gmail.com.