This document is the logbook for James Moran’s FMP. This logbook details key references to sources of information, key ideas drawn from these sources, technical and design information, discussions with my project supervisor (and any other parties involved in the project), preliminary thoughts and ideas, list of work to complete, along with plans, development of criteria for choices to be made, plus metrics for project tasks and results of technical proving exercises, experiments or technical/product evaluations.

James Moran

James Moran – FMP – Logbook

Contents

# Setting-Up a Plugin in Unreal Engine 4 (UE4)

To start off, I initiated the creation of a toolbar plugin, providing me with a base plugin, that I could build the Balanced FPS Level Generator on top of.

After attempting to have a sub-menu display to the user, via the button added to the top tool-bar of the editor, I decided it would make more sense to extend the editor’s menu listings (up to the top left corner of the default editor window).

For this, I followed a Live Training video, entitled ‘C++ Extending the Editor’, which has shown me how to add custom menu options, to the menu items list (for the ‘Edit’ sub-menu). (Noland, 2015)

This menu item will be added to the custom ‘Procedural’ category, on the ‘Edit’ sub-menu, as shown below (currently a blank box is shown, as there are no children of the UBaseEditorTool class, to be displayed in this category, from which, the Balanced FPS Level Generator will derive from):



This will then show an Edit Properties dialog for the user to choose constraints/other settings, before initiating the level generation (such as the area to generate a level in and bias of the generation for certain aspects of the level). An example of this is shown below:



# Considered Methods for Balancing the Space Filling Algorithm

First, looking at a paper, entitled ‘Procedural Generation of Balanced Levels for a 3D Paintball Game’. (Raul Lara-Cabrera et al, 2017)

In this paper, the idea of using zones, with edges between the zones (up to a certain threshold), connecting them, with a vector representation, holding three specific groups of values: First, the coordinates of the zone, then the density of obstacles in that zone and finally, the density of obstacles for edges. (Raul Lara-Cabrera et al, 2017)

In order to maintain a balanced map (level), with no clear advantage for either team, an attribute known as ‘Fitness’ is used, with coefficients for defensiveness, flanking and dispersion considered, along with the mean and standard deviation of the defensiveness and flanking values of the zones. (Raul Lara-Cabrera et al, 2017)

The defensiveness of a zone, is determined by the following factors:

* The density of the obstacles within that zone
* The density of the obstacles between that zone and the nearest zones (on the edges)

(Raul Lara-Cabrera et al, 2017)

The flanking coefficient of a zone, is calculated by counting the number of connected zones in the sub-graph, composed of the zone’s adjacent nodes, after removing the zone from consideration. If a zone has no connected zones, its flanking coefficient is zero. (Raul Lara-Cabrera et al, 2017)

For this algorithm’s genetic operators, the generator has a variation probability of Y (with a default value of 0.1) and a recombination probability of Z (with a default value of 0.75). Mutation and crossover operators are used here, with the mutation operator applying pseudo-random permutations to the values of an individual vector (as noted in the 2nd paragraph of this section), adding to or multiplying by a respective pseudo-random value. The decision of adding or multiplying is also decided upon by chance, with the same probability. If an individual vector is mutated to such an extent, that it becomes invalid for the FPSLevelGenerator’s requirements, the algorithm will assign a fitness value of zero to it. The map graph is then recalculated after this mutation, to include new edges between zones, if they become close enough because of it. (Raul Lara-Cabrera et al, 2017)

For the crossover operators, there are two of these, which are not applied at the same time, but with a probability of 0.5 for each. The first of these, is a one-point crossover (Raul Lara-Cabrera et al, 2017)

# Software Development Analysis of Classes for the Method Detailed In: ‘Procedural Generation of Balanced Levels for a 3D Paintball Game’

From this considered method for balancing the space filling algorithm, the following classes can be discerned from it:

* Area (with position coordinates, extents and area object density)
* Zone (inheriting from Area, including the zone’s edges and a ZoneFitnessValue as well)
* Edge (inheriting from Area, to connect zones that are close enough to each other)
* ZoneFitness (with coefficients for defensiveness, flanking and dispersion considered, along with the mean and standard deviation of the defensiveness and flanking values of the zones)

These can be represented in a Class Diagram (as shown below):

# Bibliography

**There are no sources in the current document.**

# References

CABRERA, R., L., *et al*, 2017. *Procedural Generation of Balanced Levels for a 3D Paintball Game*. Madrid: Universidad Autonoma de Madrid

NOLAND, M., 2015. C++ Extending the Editor | Live Training | Unreal Engine. [Viewed on the 04/03/2018]. Available from: <https://www.youtube.com/watch?v=zg_VstBxDi8&t=1364s>