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

[Considered Methods for Balancing the Space Filling Algorithm 2](#_Toc509568953)

[Setting-Up a Plugin in Unreal Engine 4 (UE4) 4](#_Toc509568954)

[Balanced FPS Level Generation System 6](#_Toc509568955)

[First Row 9](#_Toc509568956)

[Second Row 10](#_Toc509568957)

[Third Row 10](#_Toc509568958)

[Screenshots from the Implementation 11](#_Toc509568959)

[Bibliography 14](#_Toc509568960)

[References 15](#_Toc509568961)

# 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) will be utilised. A vector representation, holding three specific groups of values will be used for each zone. First, the coordinates of the zone, then the density of obstacles in that zone and finally, the density of obstacles for edges, are taken into consideration. (Raul Lara-Cabrera et al, 2017)

In order to maintain a balanced map (level), with no clear advantage for either team, coefficients for defensiveness, flanking and dispersion are used, 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)

Considerations for the flanking coefficient, are shown on the next page.

To alter how the level is generated, the generator will use a mutation-operator. The mutation operator applies 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 not consider this vector for the map. 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)

This ties in with Wang Tiles quite well, as a tile and its edges can represent a zone and its edges. A unit measurement for this project is one Unreal Unit (UU), which equates to 1 metre.

Using the Defensiveness, Flanking and Dispersion values, in an equation, to calculate an ‘Overall Compatibility’ value, comes out as follows:

(Raul Lara-Cabrera et al, 2017)

Where α, β and Γ are the coefficients for defensiveness, flanking and dispersion respectively. (µd µk) is the mean and (σd, σk) is the standard deviation, of the defensive and flanking values respectively. (Raul Lara-Cabrera et al, 2017)

For the defensiveness coefficient, di the following equation is used:

(Raul Lara-Cabrera et al, 2017)

Where the density of obstacles within that zone (Densityi), is calculated with the following equation:

(Raul Lara-Cabrera et al, 2017)

With δi equating to the object density of zone i and δMAX equating to the highest density.

For the density of the paths to and from that zone (Densitypaths), the following equation is used:

(Raul Lara-Cabrera et al, 2017)

Where γi is the density of objects in zone i and δj represents the density of objects in edge j.

For the flanking coefficient (ki), this equation is used:

(Raul Lara-Cabrera et al, 2017)

Where φi is the number of connected zones in the sub graph, created from the adjacent zones to the node being considered, not including that zone itself and γi is the number of zones connected to the node being considered. If this zone has no connected zones, its flanking coefficient is zero (for example, a zone is in between 5 other zones, with the top 2 zones only being connected to each other and the centre zone, with the bottom 3 zones only being connected to each other and the centre zone. After taking the centre zone out of consideration, there are two sets of connected zones, so ki = 1 – 2/5 = 3/5). (Raul Lara-Cabrera et al, 2017)

For the edges, edge density (δj) is used to represent the colour of an edge. This is calculated from the volume taken up by objects in the zone (in cm3), divided by the mean distance (in cm) of these objects, from the centre point of the edge. These equations are used for such:

δj is then checked against the range of values used to determine the colour of an edge. Where if

# 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:



This currently produces a structure to encapsulate the level, for the respective dimensions, with a light source (point light), at the centre of this structure. This can be seen below:

Level Generation Bounds Exterior, Unlit:

Level Generation Bounds Interior, Lit:

This provides a platform, to generate the level on top of, which is detailed in the next section.

In forming this encapsulation-object, a source from the UE4 documentation and the UE4 AnswerHub, were used for reference. (Epic Games, 2017), (Digi Labs Dan, 2016)

# Balanced FPS Level Generation System

The following diagram demonstrates the use of these equations:

(MS Paint Pixel-Based Calculations)

749: Depth (80 cm), 551: Width (60 cm)

(Average pixels to cm = ((749 / 80) + (551 / 60)) / 2 = (9.3625 + 9.1833) / 2 = 9.273 (to 3 d. p.).

(308, 21): Centre Edge Point.

Distance ((462, 175), (308, 21)) =

Distance ((220, 379), (308, 21)) =

Distance ((214, 606), (308, 21)) =

Distance ((389, 606), (308, 21)) =

As this value lies between 2500 and 5000, the edge colour for the top edge of this zone, is red.

Another example of its use, is detailed in this section. Note that no matter which algorithm is used to fill the area, the set-up shown immediately below, is always used, before the space is filled with the respective space-filling algorithm:

The width and depth for this area, are expressed by the user (in cm). This will encapsulate the volume to generate a level within (filling space from the ‘Generation Area Floor’, to the default height of the level (100cm or 1UU).

For the use of Wang Tiles, each Wang Tile (Zone class instance) in this implementation, will take up a volume of 1m3 (1x1x1 UU). Specifically, as shown in the diagram immediately to the right, is this set-up for the placement of these Zones (for a 3x3m area (9UU2)):

As there are walls encompassing this area, they are presumed to have a red-edge (matching up to the tiles).

A top-down view (to scale), of all the Wang Tiles in the set, is provided below:

There are 11 Wang Tiles in this set used by the level-generator.

The anchor point of each object on the tile is 5 cm from the edge of that object.

The calculations of the edge-colour, for each of the edges of each wang tile, is provided below (starting at the top left of the above diagram, for WangTile1, moving rightwards for the next Wang Tile, on a row by row basis, with the ranges for the edge-colour as: Where if ): (Omitted) The Geometry for Zones 9 and 11, allow the Player to pass through them, by (from the west or east to the opposite direction, over the obstacles in the Zone).

These Edge colours, are represented on the diagram of Wang Tiles, as per the calculations, as shown below (along with a number to identify each tile, when mentioned in a text passage of this document):

As per the example grid layout (shown earlier in this section), these tiles will now be placed onto the grid, making sure that they match up to the edges correctly.

If there is more than one tile available, in the set of sub-tiles, that match the current grid position, a random number will be used to decide on which tile to use from this sub-set. This has been generated by Random.org. (Random.org, 2018)

Placement of tiles will begin on the lowest row of the grid, in the first column, moving rightwards. The colour of the geometry on the tiles, has been changed to black (to have them stand out).

## First Row

First Column: Valid tiles: 2, 3, 4, 7, 8, 9 and 10. Tile chosen: 10.

Second Column: Valid tiles: 2, 3, 4, 7, 8, 9 and 10. Tile chosen: 8.

Third Column: Valid tiles: 2, 3, 4, 7, 8, 9 and 10. Tile chosen: 4.

The generation area, after selecting tiles for the first row, is shown to the left:

## Second Row

First Column: Valid tiles: 2, 3, 4, 7, 8, 9 and 10. Tile chosen: 7. Second Column: Valid tiles: 2, 3, 4, 7, 8, 9 and 10. Tile chosen: 4. Third Column: Valid tiles: 5. Tile chosen: 5. The generation area, after selecting tiles for the second row, is shown below:



## Third Row

First Column: Valid Tiles: 2, 7, 8, 9 and 10. Tile Chosen: 7. Second Column: Valid Tiles: 5. Tile Chosen: 5. Third Column: Valid Tiles: 2, 7, 8, 9 and 10. Tile Chosen: 9. The generation area, after selecting tiles for the third column, is shown to the right:

After completing these sample calculations, it has become clear that the equation for determining the edge-density of a Zone’s Edges, provides inconsistent values. This can result in output, as is shown, in the diagram to the right. This shows a level, where the Player is not able to move between all the Zones, no matter where their Spawn Point is placed in the level. Modifications to the Edge Colour calculations, will have to be undertaken to resolve this issue. Heuristics will be considered for this along with reconsidering the tile design (as in this example, the 11th tile will have never been considered, as there is no configuration of tiles that would allow for the placement of it).

# Screenshots from the Implementation

After resolving certain issues with the plugin’s use of Blueprint assets, these screenshots have been taken of the level. These are to scale (from a top-down perspective, with the floor and ceiling of the level generation area, removed so that the tile geometry can be seen):





This shows that the steps for the generation of a level, fall into place, but as this is based off the current set of logical steps, it is still subject to the issues identified in the previous section. Rules will have to be put in place, to have the generator put together a level that allows the Player to move between most of the Zones of the level. After this, factors for balancing the level can be considered.

# Improvements to the First Prototype

After reflecting on the improvements, I could add to the first prototype (as detailed in the previous section), I would want to consider the following for the next prototype (in this order):

* Using heuristics to determine the edge-colour of a tile, instead of an equation (to make sure the level is mostly traversable)
* Adding new tiles to the set of tiles to use (at least for copies of the tiles currently present in the set, but at a different orientation)
* Considering the Defensiveness, Flanking and Distribution characteristics of a Zone, along with the Heuristics, to ensure a level is balanced for both teams, as well as mostly traversable

To initiate these improvements, will be that of designing with heuristics (for a traversable level, even though, more than likely, not a balanced level).

## Level Generation Heuristics

For these heuristics, I will want them to use the Blue edge-colour, to indicate a solid wall (that is impassable by any Player). This means that the ‘virtual-edges’ of the ‘virtual-zones’ , for the walls that encapsulate the level-generation area (with the respective north, east, south or west ‘virtual-edge’), will have a blue edge-colour, as well as the edges of any Zone, which completely obstruct Player movement.

This logic will be applied to the previously, ‘lower’ edge-colours (given the range for edge-density values used to determine the edge-colour), for green equating to an edge with chokepoints (either a narrow opening between two Zones, or an obstacle that must be vaulted over), red equating to edge with a wider opening (but not completely clear, or multiple smaller openings and/or minor obstacles that the Player can walk over, experiencing a minor ascent or descent) and grey equating to an edge with no obstructions to Player movement at all (from this Zone to another).

There is also logic for how edge colours match-up to each other, as shown in this table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Edge Colour to Compare** | **Against Blue Edge (%)** | **Against Green Edge (%)** | **Against Red Edge (%)** | **Against Grey Edge (%)** |
| Blue | 82 | 6 | 6 | 6 |
| Green | 6 | 14 | 35 | 45 |
| Red | 6 | 10 | 24 | 60 |
| Grey | 6 | 6 | 6 | 82 |

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)
* Edge (that of the connection point to and from zones, that are close enough to each other)

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

DIGI LABS DAN, 2016. *Create a new Level from C++ code* [Viewed on the 19/03/2018]. Available from: <https://answers.unrealengine.com/questions/440347/create-a-new-level-from-c-code.html>

Epic Games, Inc., ©2004-2017. *UWorld* [Viewed on the 19/03/2018]. Available from: <http://api.unrealengine.com/INT/API/Runtime/Engine/Engine/UWorld/index.html>

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>

RANDOM.ORG, © 1998-2018. *What’s this fuss about true randomness?* [Viewed on the 18/03/2018]. Available from: <https://www.random.org/>