Balanced First Person Shooter Level-Generator

James Moran

# Overview

Having played various First-Person Shooters (FPS), since my early teenage years, as well as having seen the competitive play of more recent FPS titles (such as Counter Strike: Global Offensive (CS: GO)), it has become apparent to me, that creating balanced levels for FPS titles, could be considered a project in of itself. That is in a nutshell, levels that would not favour one side or the other, on an overall basis (with advantages and disadvantages for each team, given where they start in a level, that the Players must overcome). For the creation of balanced levels, having a team dedicated to such a purpose seems necessary, with them using a substantial quantity of the overall project’s resources (the game’s), to create balanced levels, that have had a fair amount of work put into them.

For context on this issue, there are Players who feel as though specific levels of certain FPS titles, favour one side far more greatly than the other. This leads to Players having feelings of unfairness, in relation to how they are at a disadvantage from the start of a game (irrespective of Player skill-level). This is in respects to the paths Players can take through the level, along with where they would have to look to find enemies at certain points in a path, or from entrances/exits to/from a path, as well as entrances to other paths. (LevelCapGaming, 2014)

The scope of the project, will be broken down into three main phases (with two bonus phases):

* Greybox Phase: In this phase, the initial framework for the level generator will be implemented, so that it can produce the geometry for the level (such as the walls, entry and exit points, as well as obstacles to provide cover)
* Collision Bounds Phase: After the geometry for the level has been generated, one could import this level, as a mesh into a game project. The problem with that is, the project handling system (e.g. a game-engine), would consider the level asset as one asset, applying a collision box or sphere, that envelops the whole level. For this phase then, the Level-Generator would have to create collision bounds for each piece of geometry it has generated in the first phase (for walls, obstacles, doorways and other entry/exit points to name a few)
* Texture Phase: After a level with suitable geometry and collision bounds has been generated, comes that of applying appropriate textures to the geometry, as per the setting of the game’s level (such as clinical, office related textures, for the corridors/cubicles and walls, of an office building)
* Bonus Phase One: Prop Phase: This is a bonus phase (a stretch goal), as it is not critical to the purpose of the project and will only receive implementation, if there is suitable time for such, after completing the first three phases of the project. In this phase, props (either dynamic or static) will be added to the level accordingly (such as chairs, desks, stationary equipment, water-coolers, given an office setting). These can either be destroyed, moved or broken through, to remove them as additional obstacles in the level.
* Bonus Phase Two: Lighting Phase: This is a bonus phase (a stretch goal), as it is not critical to the purpose of the project and will only receive implementation, if there is suitable time for such, after completing the first three phases of the project. If this phase is not undertaken in the project’s development timeline, an ambient (global) light source, will be applied evenly to the whole level, but if this phase is undertaken, then appropriate lighting for each section of the level, will be generated by the Level-Generator (coming from ceiling lights, lamps, torches, external sources (such as from windows), as well as from certain other props in the level (such as a mobile-light source, provided to one or both teams)

# Overall Aim(s)

Considering this, the overall aim of the project will be that of creating a tool, that generates a level, for an FPS, that has an interior context (such as a cave or an office building), with one main degree of level gradient (as this level will take place on, for example, one level of a building, with minor ascent possible via obstacles that the Player can get on top of, if accommodated for in the level-generator). In addition, the generator will produce a ‘balanced’ level, that takes account of various properties for a balanced level, such as movement paths through the level (to objectives or otherwise), choke-points in the level, as well as where Players will want to check for enemies, given the route they have taken through the level. This will be implemented as either an engine plugin for the Unity or Unreal Engine 4 (UE4) game-engines, or as a stand-alone native-C++ application.

# Initial Objectives

When thinking of some initial objectives for the project, the following come to mind:

* Consider relevant literature, for algorithms that one could utilise for (procedural) level-generation (literature considered for review and initial lines of research, will be listed in the next section)
* Using the chosen level-generation methodology, implement this method to the extent that it will generate the level’s ‘floor’ (to be used as a basis for all other parts of the level)
* Following on from the previous point, implement functionality to allow the system to generate bounds, surrounding the base floor of the level (such as a wall on each of the four sides, to represent a level of a building)
* After this, then move onto to implementing functionality to allow the system to generate the main geometrical features of the level, contained within the bounds of the level (such as rooms with their subdivisions for a building)

# Relevant Literature

I intend to use the following sources to guide me in the implementation of the project (This is subject to change):

Procedural Content Generation in Games (Computational Synthesis and Creative Systems) – Noor Shaker, Julian Togelius and Mark J Nelson.

This book presents the most up-to-date coverage of procedural content generation (PCG) for games, specifically the procedural generation of levels, landscapes, items, rules, quests, or other types of content. Each chapter explains an algorithm type or domain, including fractal methods, grammar-based methods, search-based and evolutionary methods, constraint-based methods, and narrative, terrain, and dungeon generation.

The authors are active academic researchers and game developers, and the book is appropriate for undergraduate and graduate students of courses on games and creativity; game developers who want to learn new methods for content generation; and researchers in related areas of artificial intelligence and computational intelligence.

(Springer International Publishing AG, © 2017)

This source will provide me with the relevant theories on procedural level generation, which I can then use as a basis for level generation in this project.

Level design: Processes and experiences – Christopher W. Totten.

In this book, veteran game developers, academics, journalists, and others provide their processes and experiences with level design. Each provides a unique perspective representing multiple steps of the process for interacting with and creating game levels – experiencing levels, designing levels, constructing levels, and testing levels. These diverse perspectives offer readers a window into the thought processes that result in memorable open game worlds, chilling horror environments, computer-generated levels, evocative soundscapes, and many other types of gamespaces. This collection invites readers into the minds of professional designers as they work and provides evergreen topics on level design and game criticism to inspire both new and veteran designers.

(CRC Press, ©2017)

This source will offer me the prerequisite knowledge, for developing an engaging level, that I can then use as a basis, for the properties of a level, that this level-generator must adhere to.

The Science of Level Design: Design Patterns and Analysis of Player Behaviour in First-person Shooter levels – Kenneth Hullett

ABSTRACT

Kenneth Hullett

The Science of Level Design: Design Patterns and Analysis of Player Behavior in

First-Person Shooter Levels

Level designers create gameplay through geometry, AI scripting, and item placement.

There is little formal understanding of this process, but rather a large body of design

lore and rules of thumb. As a result, there is no accepted common language for

describing the building blocks of level design and the gameplay they create. This

dissertation presents a set of level design patterns for first-person shooter (FPS)

games, providing cause-effect relationships between level design patterns and

gameplay. These relationships are explored through analysis of data gathered in an

extensive user study.

This work is the first scientific study of level design, laying the foundation for further

work in this area. Data driven approaches to understand gameplay have been

attempted in the past, but this work takes it to a new level by showing specific cause

effect relationships between the design of the level and player behavior.

The result of this dissertation is a resource for designers to help them understand how

they are creating gameplay through their art. The pattern collection allows them to

explore design space more fully and create richer and more varied experiences.

(Kenneth M. Hullett, 2012)

This source will provide me with an in-depth level of detail, into specific design patterns for the levels of an FPS, given extensive detail on the many components to consider in the level of an FPS.

# Project Management Approach

The Project will use standard project-management techniques. This process is initiated by identifying certain characteristics of the Project, these are:

* The Project’s objective
* When the Project is to have been completed by (scheduling)
* Project Complexity
* Tasks of the Project, the time required to complete these tasks and how one should complete a project task
* Available Resources
* Organisational Structure
* Information and Control Systems

(James Moran, 2017).

The techniques used to manage the project, are most notably that of the Program Evaluation Review Technique (PERT) Charts and Gantt Charts, for suitable time management of the Project, to keep on track with the project, not finishing tasks too late or too early (James Moran, 2017).

The Software Development Methodologies considered for the project, are that of the Spiral Methodology, the Agile Methodology and the Feature Driven Development Methodology. A respective overview of each methodology is as follows: (Spiral Methodology)

The Spiral Lifecycle Model is a sophisticated lifecycle model that focuses on early identification and reduction of project risks. A spiral project starts on a small scale, explores risks, makes a plan to handle the risks, and then decides whether to take the next step of the project - to do the next iteration of the spiral. It derives its rapiddevelopment benefit not from an increase in project speed, but from continuously reducing the projects risk level - which has an effect on the time required to deliver it. Success at using the Spiral Lifecycle Model depends on conscientious, attentive, and knowledgeable management .It can be used on most kinds of projects, and its risk-reduction focus is always beneficial.

The spiral methodology extends the waterfall model by introducing prototyping. It is generally chosen over the waterfall approach for large, expensive, and complicated projects.

At a high-level, the steps in the spiral model are as follows:

1. The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.

2. A preliminary design is created for the new system.

3. A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.

4. A second prototype is evolved using four steps:

Evaluate the first prototype and identify its strengths, weaknesses, and risks.

Define the requirements of the second prototype.

Plan and design the second prototype.

Construct and test the second prototype.

5. At the project sponsor's option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could result in a less-than-satisfactory final product.

6. The existing prototype is evaluated in the same manner as was the previous prototype, and, if necessary, another prototype is developed from it according to the fourfold procedure outlined above.

7. The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.

8. The final system is constructed, based on the refined prototype.

9. The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large-scale failures and to minimize downtime. (IT Knowledge Portal, 2017)

(Agile Methodology)

Agile Software Development Methodology

Agile software development is a conceptual framework for undertaking software engineering projects. There are a number of agilesoftware development methodologies e.g. Crystal Methods, Dynamic Systems Development Model (DSDM), and Scrum.

Most agile methods attempt to minimize risk by developing software in short timeboxes, called iterations, which typically last one to four weeks. Each iteration is like a miniature software project of its own, and includes all the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation. While iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration. At the end of each iteration, the team reevaluates project priorities.

Agile methods emphasize realtime communication, preferably face-to-face, over written documents. Most agile teams are located in a bullpen and include all the people necessary to finish the software. At a minimum, this includes programmers and the people who define the product such as product managers, business analysts, or actual customers. The bullpen may also include testers, interface designers, technical writers, and management .

Agile methods also emphasize working software as the primary measure of progress. Combined with the preference for face-to-face communication, agile methods produce very little written documentation relative to other methods. (IT Knowledge Portal, 2017)

(Feature Driven Development Methodology)

Jeff De Luca and Peter Coad were both greatly involved in developing the Feature Driven Development methodology. Peter describes FDD as having just enough process to ensure scalability and repeatability while encouraging creativity and innovation.

More specifically, Feature Driven Development asserts that:

A system for building systems is necessary in order to scale to larger projects.

A simple, but well-define process will work best.

Process steps should be logical and their worth immediately obvious to each team member.

“Process pride” can keep the real work from happening.

Good processes move to the background so team members can focus on results.

Short, iterative, feature-driven life cycles are best.

FDD proceeds to address the items above with this simple process (numbers in brackets indicate the project time spent):

1. Develop an overall model (10 percent initial, 4 percent ongoing)

2. Build a features list (4 percent initial, 1 percent ongoing)

3. Plan by feature (2 percent initial, 2 percent ongoing)

4. Design by feature

5. Build by feature (77 percent for design and build combined)

(IT Knowledge Portal, 2017)

The project management tools that I will use, are that of Microsoft Project, to construct PERT and Gantt Charts, along with Microsoft Visio, for creating Flow Diagrams/UML Activity Diagrams, UML Class Diagrams etc.

A Trello board, will also receive construction for the project, to make sure that certain important tasks, receive the necessary priority required to conduct the necessary sub-tasks for the project.

Along with using a Trello board, source control for the project’s source files will also be utilised. The GIT source control system has been chosen for this, with a sub-system that uses GIT (such as GitHub and GitKraken).

# Initial Plan

## Risk Assessment and Evaluation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Risk Description | Probability of Occurrence (%) | Severity (1-10, 1 = negligible, 10 = catastrophic) | Loss Size (Days) | Risk Exposure (Probability x Loss Size) | Priority (Probability x Severity) | Contingency Plan |
| The Software Development Methodology that was chosen, is found out to be unsuitable for the project. |  |  |  |  |  |  |
| Following (sample) end-user testing additional features are requested (that were not foreseen). | 70 | 2 | 6 | 4.2 | 1.4 | Put in place a modular system, composed during the initial planning phases, that allows for addition of features to the generator, on an ad-hoc basis (fluidly). |
| Requirements are found to have not received full definition. | 55 | 4 | 10 | 5.5 | 2.2 | Make sure to follow the requirements gathering process thoroughly, so as to reduce the imprecision of any requirement definitions, if any imprecision is identified. |
| The project’s deliverables are not finished in the time that was calculated, for how long it should take to finish them. | 50 | 5 | 10 (overtime) | 5 | 2.5 | Making sure to utilise any spare time as effectively as possible (if completing other parts of the project before they are due), as well as allotting suitable leeway, to the time it should take to finish a component of this project, accounting for any delays. |
| Following (sample) end-user testing, more effort on the user guide is required. | 40 | 3 | 4 | 1.6 | 1.2 | Make sure the user guide thoroughly details all aspects of the generator, as well as the implementation of it in one’s project. |
| Software Development Methodology (SDM) deemed insufficient. | 35 | 4 | 20 | 7.0 | 1.4 | Ensure that the SDM utilised meets the expected development practices, for the generator, considering as many conditions as possible. |
| The project enters an ‘over-budget’ state. | 25 | 6 | 18 | 4.5 | 1.5 | Making sure to accurately identify costs during the planning phases, as well as having an emergency company capital funds account. |
| A power cut occurs during compilation time. | 0.1 | 9 | 40 | 0.04 | 0.009 | Making sure to compile and save as often as possible, as well as backing up the files in multiple locations. |
| A hacker is able to hack into the development system as well as any backup locations (accessible online) and corrupt/delete the project files. | 10^-9 | 7 | 7 | 0.00000000007 | 0.00000000007 | Keep an up-to-date copy of the project on a storage medium that is not connected to the internet. |
| Thieves are able to break into the location where a development platform and any physical backup storage devices are kept and steal them. | 10^-5 | 5 | 14 | 0.0000014 | 0.0000005 | Keep an up-to-date copy of the project on a cloud storage system (which is hence, not possible to ‘steal’ physically). |

(James Moran, 2017).

# References

1. LevelCapGaming, 2014. *FPS Level Design – LevelCap Bashes BF4 Map Design* [viewed 02/12/2017]. Available from: <https://www.youtube.com/watch?v=FN7iLKUR8eY>
2. James Moran, 2017. CGP504 – Project Development and Project Preparation. Southampton Solent University, CGP504, BSc Computer Games (Software Development), 2nd Year. Submitted online via Solent Online Learning. Available at: <https://learn.solent.ac.uk/>
3. ©2017 Springer International Publishing AG. Procedural Content Generation in Games [viewed on the 05/12/2017]. Available from: <http://www.springer.com/gb/book/9783319427140>
4. ©2017 CRC Press, Taylor & Francis Group. Level Design: Processes and Experiences [viewed on the 05/12/2017]. Available from: <https://www.crcpress.com/Level-Design-Processes-and-Experiences/Totten/p/book/9781498745055>
5. Kenneth M. Hullet, 2012. *The Science of Level Design: Design Patterns and Analysis of Player Behavior in First-Person Shooter Levels*. University of California, Santa Cruz: University of California [viewed on the 05/12/2017]. Available from: <https://users.soe.ucsc.edu/~ejw/dissertations/Ken-Hullett-dissertation.pdf>
6. IT Knowledge portal, 2017. Software Development Methodologies [viewed 09/12/2017]. Available from: <http://www.itinfo.am/eng/software-development-methodologies/>