Third Year Project: Literature Review

Matthew Taylor

Registration: 100151729

1 Introduction

1.1 Aim of the project

The project aims to implement a first person shooter game which uses procedural generation to provide a novel and interesting gaming experience. There are many ways procedural content generation (PCG) can be applied to games, so this literature review will explore the broad themes of the field to find specific areas to focus on.

1.2 Areas of knowledge required

PCG techniques have been developed and used in video games since the 1970s. I will need to gain a strong understanding of this field in order to find interesting areas to focus on.

In order to implement the concepts and techniques I will focus on, I will need to learn how to use the Unity platform.

1.3 Report structure

The report will provide a summary of the PCG field, identifying key themes and the current state of the field, including interesting "unsolved problems" or under-explored areas.

I will then focus on a few promising areas to explore the literature in detail, with a view to informing my project's aim and objectives.

This report will critically analyse sources as they are referenced. This will inform my conclusions and amendments to my project's aim and objectives, which I will present at the end of the report.

2 Overview of Procedural Content Generation

2.1 Types of Procedural Content Generation

There are two main types of procedural content generation, which use similar methods but achieve different goals. They are **endless content generation** and **efficient content storage**.

2.1.1 Efficient content storage

Prevalent in the early days of PCG, efficient content storage exploits procedural content generation to create large amounts of data without needing to store it - it can simply be re-generated using a deterministic algorithm every time it is required. An example

of this is the 1984 game *Elite*, as documented in Spufford (2003). Here, the game developers discuss using a pseudorandom number generator with pre-defined seeds to generate large amounts of content, without needing to store any of the content itself.

A similar technique is adopted for image generation, as per Perlin (1985). Perlin describes using well-defined stochasic functions that can be parametised to produce realistic textures procedurally. This means that textures can be generated when required or when a program is initialised, rather than needing to store them. Perlin's paper is widely cited and had a large impact on the field, despite it's relative age.

A lot has changed in computers since the 80s - memory and storage constraints are now much less of an issue, so using PCG to efficiently store large amounts of content has been used less.

2.1.2 Endless content generation

As storage constraints have been eclipsed by the rapid development of technology, the generation of content has become more of a concern. Increasing sophistication of games and the expectations of players mean content creation is more time-consuming and costly. This has resulted in an increase in the use of PCG to generate content.

This has become a wide field, resulting in the creation of a taxonomy of PCG in Hendrikx et al. (2013). It describes six layers of PCG in games:

1. Game bits (eg. textures, buildings)
2. Game space (eg. maps, lakes)
3. Game systems (eg. roads)
4. Game scenarios (eg. puzzles, stories)
5. Game design (eg. rules)
6. Derived content (eg. leaderboards)

This paper is relatively recent (2013) which makes the summary it provides still accurate to this day. It is cited (36 times on acm.org) by others in the industry. I have done some comparison against http://pcg.wikidot.com/ which maintains a list of games using PCG, which confirms there have been no large advances or practical examples of PCG in games since this was published. This gives me confidence that the taxonomy and techniques discussed are still accurate and relevant today.

2.2 Areas of PCG to focus on

In chapter 6 of Hendrikx et al. (2013), the paper discusses "recommendations for future research". One of the recommendations details the generation of realistic indoor game spaces. It is also clear from the literature overview that many of the applications of PCG happen before game play begins, so I will be interested to explore the applications of real-time PCG.

In the following section, I will narrow my focus to the area of generating indoor spaces, in terms of both layouts and decoration, especially in real-time.

3 Procedural generation of indoor spaces

3.1 Room generation

3.1.1 Agent based approach to decoration

Germer and Schwarz (2009) describe an agent based approach to decoration, which generates the furniture placement in a room. Agents are assigned to particular objects and, when entering a search state, seek positions that meet particular constraints and goals. It is designed to be deterministic (using PRNGs) and to generate rooms at speed (to avoid storing lots of geometry), when required. These are interesting properties that I would like to explore further. The authors acknowledge that high-level control and sophistication of the generated layouts are lacking, so these are areas I could look to build on.

This paper is now relatively old (2009) and not particularly well cited (22 citations on Wiley Online Library). It uses relevant techniques (agent based search) but uses out-dated technology (VRML). The ideas in the paper are interesting and described in considerable detail, so it is still a useful paper to consider.

3.1.2 Computerised clutter

Taylor and Parberry (2010) discuss "computerised clutter" and the challenges involved in making a procedurally generated room look realistic. They list five properties that are desirable in a room generator: novelty, structure, interest, speed and controllability. The paper describes in detail how to procedurally generate "clutter" in a room which does not look computer generated, by using the concept of "anchor points". These points are adjusted with gaussian noise to provide natural looking variation around points and patterns that can be designer-adjusted, making them look human-constructed when in reality they are generated by computers.

This paper is very specific although not well cited (1 citation on researchgate.net). It is also not very recent, published in 2010. These facts mean I am not going to read

too much into the paper, especially as they primarily present only one technique of generating "computerised clutter". However, their subjective summary of what makes procedural generation of clutter in a room look convincing is useful and compelling enough that I have included it. I will incorporate some of these techniques into the decoration of my room generator.

3.1.3 Combining human and procedural content generation

Discussed in the papers above is the idea of content "looking" procedurally generated. Indeed, in the most recent prominent example of a game making extensive use of PCG - No Man's Sky (2016) - is criticised in many reviews, with Welsh (2016) as an example, for being vast but samey. To quote the review: "Some disappointment is inevitable once you've visited a handful of planets and started to see the limits of the algorithm. The parameters of weirdness in No Man's Sky have been quite strictly defined, and it gets repetitive."

Rogue-like games tend to suffer less from this phenomenon. A recent example that has been explored in detail is the game Spelunky. As detailed in Brown (2016), Spelunky uses a simple path-finding algorithm, a tile-based approach to generating levels and - crucially - combines human crafted tiles with scope for procedural generation within those tiles.

This generates detailed, interesting levels that feel unique, interesting and always have a route from beginning to end. The combination of taking human-designed tiles or areas and integrating procedural elements within them, to provide environments that feel created but are actually generated is intriguing and something I would like to explore further with the project.

4 Level generation

Hendrikx et al. (2013) in Chapter 3 discuss common methods of PCG. In relation to how indoor spaces are generated, they discuss how common methods are Pseudo-random Number Generators (PRNG) and Generative Grammars (GG).

These methods are well suited to producing feasible indoor spaces required in games. Although Hendrikx et al. (2013) detail other techniques, like genetic algorithms and fractal spatial generation, these are less common in games. I suspect this is because the output of layouts they produce, despite being procedurally generated, will either follow a discernable pattern (in the case of fractal algorithms) or will be "too random" (in the case of genetic algorithms) to appeal to human players. My aim is to create a procedural generator that feels realistic in order to be compelling and interesting to the player, so I feel these approaches may not be suitable.

4.1 Generative grammars

Dormans (2010) discusses using generative grammars to create complex, coherent, branching mission and level structures. As the article details, these generate levels more appropriate for action adventure games (as compared to levels generated by roguelikes for roleplaying games) where a pattern and structure to the level is important. The article also details the difference between mission and space.

The paper is reasonably well cited (22 citations on acm.org) and was published in 2010. The techniques it describes are certainly not dated and could concievably be used to generate content in games today.

The ideas presented are intriguing, as they specifically reference the type of game I aim to create - a 3D action-adventure style game. The use of GGs will enable sequences of rooms to be generated in which the order matters (for instance, in a maze, a beginning must be connected to a goal position). In this way, it will also be possible to procedurally generate unpredictable layouts that are nevertheless perfectly valid, in an efficient manner.

4.2 Graph based approaches

Van der Linden et al. (2013) extends the approach of using generative grammars. They recognise that previous work, especially Togelius et al. (2016), does not focus much on gameplay-based control over the procedural generation. They seek to demonstrate that by using grammars to define constraints, then using graphs to generate constrained content allows a designer to have a hand on the content generated. This is especially useful in the context of games, because it allows generated content to be directed towards something. When things like "realism" and "not looking too computer generated" are considered, this is an interesting concept.

Another interesting point they raise is methods of testing the generated content. They use histograms to measure complexity and "danger", which allows them to adjust the constraints to achieve the desired results. They also use histograms to measure the generation speed, which is something I will be focussing on. This is a useful technique and something I will aim to emulate against my own ideas.

This paper is more recent than some of the others I have reviewed and cites some of the earlier papers. They specifically address some shortcomings in those papers. This gives me confidence in the areas of the other papers that did not identify as being weak, while also providing useful insight into more modern techniques in PCG.

5 Conclusion

Through my search of the literature, it is apparent that much of the research and commercial interest in PCG has been for pre-generating content. That is, PCG is used to generate things generally ahead of time and very little in real time. I am interesting in exploring the real-time aspects of PCG, specifically in relation to level layouts of indoor spaces.

This connects with Hendrikx et al. (2013) that indoor space generation could be explored further, as well as the view in Shaker et al. (2016) that PCG could be used as a more "central" element to a game, rather than just a method to generate content. I will make my project aim to explore the idea of real-time layout generation as a fundamental part of the gameplay - something that the game could not exist without.

I will explore the following aspects of procedurally generating (and re-generating) level layouts at runtime:

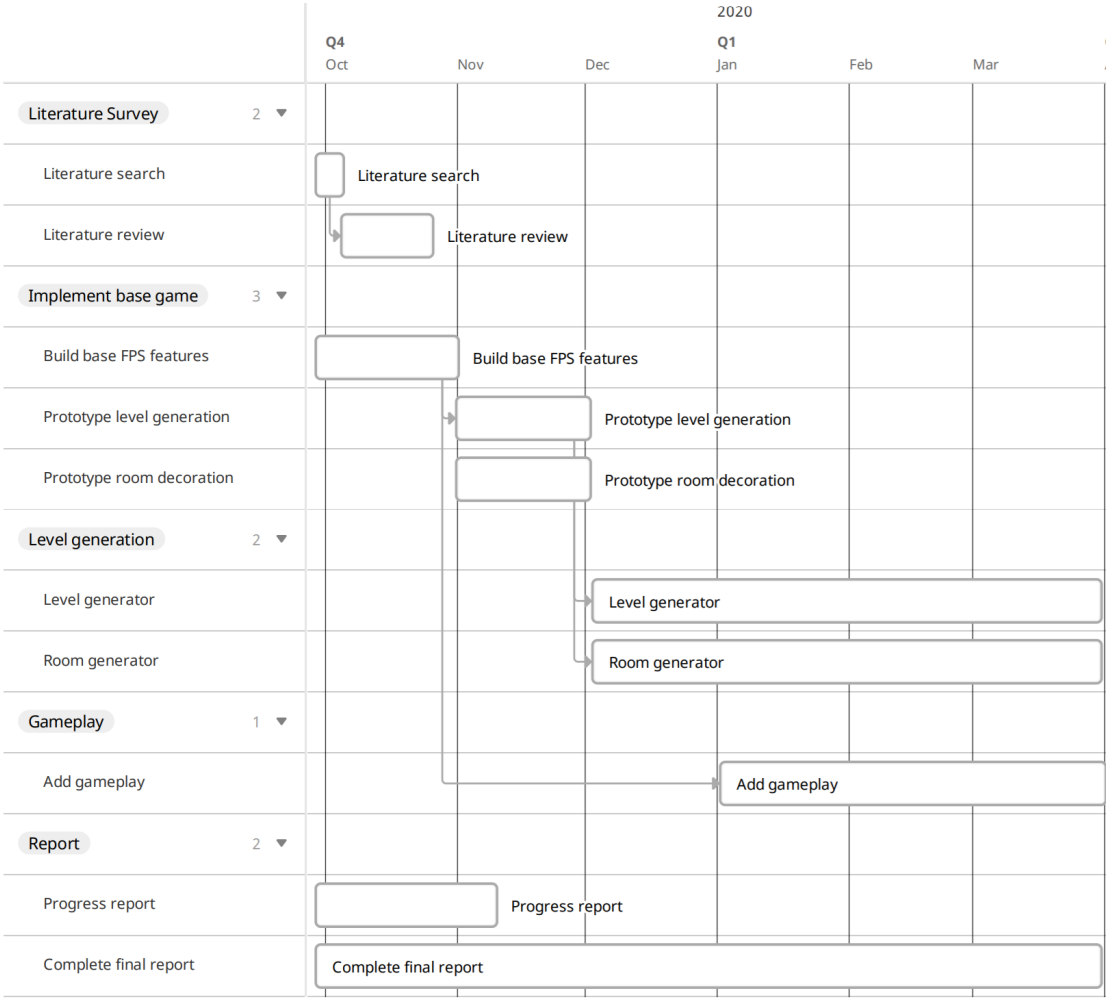
- Performance of the game
- Fidelity of generated rooms (eg. level of detail that can be generated)
- AI pathfinding in shifting layouts
- Maintaining valid level states (eg. a game that can always be completed, no matter how much the layout changes)
- Making a fun game

5.1 Changes to aims and objectives

I will change the focus of my project to look more closely at PCG. While I will maintain the initial aim of a game that changes the level layout while it is being played, I will explore this is considerably more detail, especially around the points detailed above.

I will shift focus from AI to PCG of both level layouts and the layout of furniture and decoration in the rooms generated. These are two fields which I believe could be explored in more depth. This is especially true in a run-time context, as very little literature has explored this.

5.2 Revised gantt chart



5.3 Changes to development approach

I will also adapt my development approach to more closely align with Agile principles. My timeline will be planned according to the Gantt chart above, but I will run in two week sprints.

I will use the sprint review/retrospective time to check my progress against the expected timescales, as well as making notes about performance. Where possible, I will also aim to get user feedback on the general appearance of the generated levels and rooms, in order to gain some insight into their quality.

References

Brown, M. (2016). How (and why) spelunky makes its own levels | game maker's toolkit.

Dormans, J. (2010). Adventures in level design: Generating missions and spaces for action adventure games. pages 1:1–1:8.

Germer, T. and Schwarz, M. (2009). Procedural arrangement of furniture for real-time walkthroughs. *Computer Graphics Forum*, 28(8):2068–2078.

Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1):1:1–1:22.

Perlin, K. (1985). An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296.

Shaker, N., Togelius, J., and Nelson, M. J. (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Spufford, F. (2003). *Backroom Boys: The Secret Return of the British Boffin*. Faber and Faber.

Taylor, J. and Parberry, I. (2010). Computerized clutter: How to make a virtual room look lived-in.

Togelius, J., Yannakakis, G. N., Stanley, K. O., and Browne, C. (2016). Search-based procedural content generation.

Van der Linden, R., Lopes, R., and Bidarra, R. (2013). Designing procedurally generated levels.

Welsh, O. (2016). No man's sky review (eurogamer.net).

Literature review

Introduction: brief description of project, areas of knowledge required, roadmap	First	2.1	2.2	3	Fail
Discovery of suitable quantity and quality of material	First	2.1	2.2	3	Fail
Description of key issues and themes relevant to the project	First	2.1	2.2	3	Fail
Evaluation, analysis and critical review	First	2.1	2.2	3	Fail

Quality of writing

Clarity, structure and correctness of writing	First	2.1	2.2	3	Fail
Presentation conforms to style (criteria similar to conference paper reviews)	First	2.1	2.2	3	Fail
References correctly presented, complete adequate (but no excessive) citations	First	2.1	2.2	3	Fail

Revised Workplan (if applicable)

Measurable objectives : appropriate, realistic, timely	First	2.1	2.2	3	Fail

Comments

Supervisor: Dr Rudy Lapeer

Markers should circle the appropriate level of performance in each section. Report and evaluation sheet should be collected by the student from the supervisor.