Procedural Aesthetics

Building a toolset of aesthetic devices for generative games design

*20502958*

*usw-135W300CF*

*BA (Hons.) Computer Games Design*

*2014*

*8,725 words approx.*

Table of Contents

**Introduction4**

**Chapter 1: Persistent Motivations7**

**Chapter 2: Generative Systems in Games13**

Example 1: 3D world generation in Minecraft13

Example 2: Interesting challenges in Spelunky14

Example 3: Board game layout in Advanced Heroquest15

Example 4: Card-based rules system in Fluxx17

Example 5: Character design & behaviour in The Sims18

Example 6: Procedural help/tutorial in Cloudberry Kingdom19

Example 7: Random encounters in Final Fantasy20

Example 8: Progression & difficulty in Desktop Dungeons21

Example 9: Random quests in FTL: Faster Than Light22

Example 10: Generative music in Proteus23

Conclusions of analyses24

**Chapter 3: A Rogue Gallery27**

1. The original game27
2. Early variations29
3. The Berlin Interpretation31
   1. High value factors31
   2. Low value factors33
4. Modern emergence34

**Chapter 4: Mathematical Aesthetics36**

1. Chaos37
2. Fractal sets39
3. L-systems40
4. Perlin noise43
5. Cellular automata43
6. Golden ratio44
7. Fibonacci sequence45
8. Maze and path-finding algorithms46

**Chapter 5: Modular Patterns47**

1. Spelunky template example49
2. Modular assets in The Sims50

**Conclusions54**

**References58**

**Games References59**

**Figure References60**

# Table of Figures

[Figure 1: Minecraft Landscape 13](#_Toc378198880)

[Figure 2: Spelunky - difficult challenges 14](#_Toc378198881)

[Figure 3: Advanced Heroquest - Modular board layout 15](#_Toc378198882)

[Figure 4: Advanced Heroquest - Custom-designed level layout 16](#_Toc378198883)

[Figure 5: Fluxx cardtypes 17](#_Toc378198884)

[Figure 6: The Sims (3) - Choosing character behaviours 18](#_Toc378198885)

[Figure 7: Cloudberry Kingdom - A procedurally generated 'hint-path' 19](#_Toc378198886)

[Figure 8: Final Fantasy (9) - Wandering in the wilderness 20](#_Toc378198887)

[Figure 9: Final Fantasy (9) - Random battle encounter 20](#_Toc378198888)

[Figure 10: Desktop Dungeons - Optimal mob spawning for progression 21](#_Toc378198889)

[Figure 11: FTL - A quest event is encountered 22](#_Toc378198890)

[Figure 12: FTL - Quest marker shown on star map 22](#_Toc378198891)

[Figure 13: Proteus - Procedural world; entirely generative soundtrack 23](#_Toc378198892)

[Figure 14: Rogue as it looked on a UNIX terminal 27](#_Toc378198893)

[Figure 15: NetHack's visuals 29](#_Toc378198894)

[Figure 16: Bifurcation diagram 36](#_Toc378198895)

[Figure 17: Mandelbrot figure 38](#_Toc378198896)

[Figure 18: Coastline data generated using fractal algorithms 38](#_Toc378198897)

[Figure 19: Plant forms generated using l-systems 40](#_Toc378198898)

[Figure 20: 3D tree designs formed from l-systems 40](#_Toc378198899)

[Figure 21: SpeedTree software… 41](#_Toc378198900)

[Figure 22: 3D iterations of Cantor's Dust algorithm 41](#_Toc378198901)

[Figure 23: An arrangement of Koch Snowflakes 42](#_Toc378198902)

[Figure 24: Basic Perlin noise 42](#_Toc378198903)

[Figure 25: Reticular simplex noise forming clouds 42](#_Toc378198904)

[Figure 26: John Conway's Game of Life 43](#_Toc378198905)

[Figure 27: A line depicting the golden ratio 44](#_Toc378198906)

[Figure 28: Fibonacci spiral 45](#_Toc378198907)

[Figure 29: Aloe spiral 45](#_Toc378198908)

[Figure 30: A bit-grid demonstrates… 47](#_Toc378198909)

[Figure 31: Super Mario Bros. 48](#_Toc378198910)

[Figure 32: Pac-Man 48](#_Toc378198911)

[Figure 33: Spelunky encoding 50](#_Toc378198912)

[Figure 34: The Sims (original) - modular character customisation 51](#_Toc378198913)

[Figure 35: The Sims (4) – body types 51](#_Toc378198914)

[Figure 36 (a, b & c): The Sims (2) – modular textures 52](#_Toc378198915)



# Introduction

#### Preamble

Almost every videogame at some point is bound to include some product of a random generator – whether it is to give your enemy a lucky, fighting chance against your well-practised attacks, a system to shake up a strategy game, or a world of random topography and surprising monsters. The origins of procedurally generated environments and in particular infinitely expansive worlds came about as a product of technical determinism from an optimising solution. How is it then that after a few decades of rapid technological advancement, with the need for such optimisations all but eliminated, are games with significant usage of procedural generation elements still some of the most successful and popular?

Before we get started I must point out that the two terms used in this paper’s title are *tricky* and thus I will take a moment to attempt to disambiguate these concepts. Literally, the term procedural generation can be taken to mean: ‘generated using a processor’, which could just as easily be taken to describe an entirely pre-designed experience being presented to the player by the computer’s processor as it can be used to mean ‘aspects generated at random’; the latter of which is the more common understanding within the field of video games. Having made this distinction it is important that I point out that while the scope of this paper will focus mainly on those random generation and selection functions it will in no way be limited to this and will very much incorporate the analyses of many types of procedural generation systems. The terms ‘procedural’ and ‘generative’ will be used somewhat interchangeably throughout this research; I personally prefer the more descriptive latter of the two terms as I intend my focus to be more on those systems which use algorithms to produce a multitude of results rather than routines that simply perform a single or directed mathematical function such as the difference between a landscape generation routine (see Minecraft analysis, chapter 2.1) and a fractal set algorithm (see Fractals, chapter 4.2).

The other term used in the title is one which is very often misinterpreted within the field of games design and I will rely here on the excellent comparison made by the *Extra Credits* team (Portnow & Floyd, 2012), to distinguish between the concepts of aesthetics and graphics. In their video essay they discuss the popular confusion that graphics and aesthetics are one and the same, stipulating that graphics are one of the many tools used to construct the visuals of a game and that a good looking game is such despite and not because of graphical fidelity. In my own distinction I would offer the analogy that graphics are to aesthetics as branches are to trees and thus, going forward, my research will analyse many of the constituent parts: graphics, sound, music, mechanics, narrative style and other visual design choices, which make up the aesthetics of the example games’ designed experiences and emotional contexts.

The aim of this research project is to identify a set of useful tools afforded to games designers by procedural functions and the research will arrive at its conclusion: a toolset of design elements which can inform games designers working with generative systems, through a discussion of procedural rhetoric and the motivations behind generative systems, detailed analyses of example games which use them to a lesser and greater degree. This will be followed by a thorough breakdown of the *roguelike* paradigm, a genre which emerged from an outstanding procedurally generative game, and a look at various natural mathematical functions which can be levied by games designers. Lastly as a pre-cursory summary, the final chapter will be a programmatic description of various methodologies based on the findings of these analyses and an exploration of reference texts and academic theses. To complete the definition of this paper’s scope I will finally mention that during the research the analyses will not be confined to computer games – I intend to use a variety of examples of trans-media games and indeed other art-forms to describe and identify useful tools for modular, generative games design.

# 1: Persistent Motivations

*“282,000,000,000,000 galaxies to discover!” – Claim from Elite advert.*

In his book, *Persuasive Games*, Ian Bogost (2007) sets forth a new form of critical language which he calls *procedural rhetoric* that is intrinsically tied into the computational persuasion functions of videogames. I make use of this structure in my critical analyses, expanding the scope of its usage to other forms of non-digital games which rely on procedural aspects that are not necessarily computational in nature. As software in general and specifically videogames seek to emulate pre-digital mechanisms it seems natural to begin this exploration at the point where board games and other non-digital games start to become complex, looking at how generative systems can and have started to emerge from existing random mechanics. This chapter analyses a history of generative systems used in both non-digital and videogames, then discusses what motivated the emergence of procedural generation in the first place before pointing to some reasons as to why these systems persist in modern videogames.

Modern board games, with very few exceptions, all make use of random elements at play involving dice-rolls, card-drawing or spinning mechanisms; but what about randomisation at the game’s setup phase? Executing elements of luck in this way before the start of play still integrates the notion of strategy and luck, but this shift in the dynamic can often allow for a more strategic gameplay experience, relying less on luck as the game progresses and more on the concept of using resources and opportunities provided by the system to a winning advantage. Another obvious implication of these kinds of generation systems is the expansion of the notion of exploration: if elements of the game’s layout are procedurally generated, the player can be rewarded with the excitement of new discovery or the challenge of adapting to random events within play.

There are examples of games which use the placement of tiles or other such elements to generate a map, either as a pre-cursor to strategic movement, such as turn-based deployment of scenery items prior to playing a table-top war-game (Priestley, 1987, p. 8-9) or as the strategic moves themselves in such examples as *Black Path Game* (Black, 1960) and *Tsuro* (McMerchie, 2004). As these examples are in themselves strategic decisions they do not fully satisfy the random aspect of this discussion’s title, but are important to note as possibly a developmental step toward the concept of random environment generation. Equally as important and even more noteworthy is the notion of random resource allocation, such as territory cards being dealt at random before a game of *Risk* (Lamorisse, 1959) to determine initial occupation (this mechanic is optional in the original rule-set, often out-favoured by turn-based decisions for the deployment of troops). A good example of random map-generation and resource distribution in a pre-digital game exists in Steve Jackson’s board-game adaptation of his ‘Choose Your Own Adventure’ book: *Warlock of Firetop Mountain* (Jackson, 1986); in which the majority of the board is a fixed design with spaces for tiles containing events, characters and resources such as treasure to be placed at random, face-down, before the game starts. The final section of this game board contains larger spaces intended for maze tiles – modular designs which are secretly revealed to each player individually upon their encounter, proportioning dead-ends and safe passages to the goal area of the map.

As the computer is capable of thousands of times more calculations and data management than any non-digital game could propose, it is clear to see that digital developers have a much larger scope for detail in environment generation. The games which will be subsequently examined all operate systems just like that which have been identified in the non-digital examples, being afforded a greater amount of detail and variety. Additional to the motive of extending the range of variation and number of elements that can be generated; the computerisation of such systems has another need for randomly generated environments and resources. System memory and storage capacity have always imposed limits on the scale of digitally created worlds, and generating entities as they are needed in a random fashion serves to reduce greatly the overheads of any development. This is coupled with an extremely useful feature built into most computer-based random number generators which is the ability to generate a completely predictable array of “random” numbers using a seed value, allowing an exactly prescribed environment and configuration of other elements to be generated from a tiny datum. An extreme example of this optimisation is evident in the original design for *Elite* (Braben & Bell, 1984), which was intended to contain 248 galaxies (Spufford, 2003, p. 149-151). The notion of using a seeded, pseudo-random sequence generator was subsequently developed by Braben and Bell using a pair of numbers from the Fibonacci sequence (see chapter 4.7). In this way they found they could not only produce a gigantic, seemingly random star field, but that this data was wholly predictable and so the final release of Elite contained a procedurally generated universe that was persistent for every copy of the game.

Three discreet effects pertaining to the reception of generative systems by the player have been identified by Noah Wardrip-Fruin in his book, *Expressive Processing* (2009, p. 15-16) which can be used to predict the user experience and thus curtail efforts toward building appealing aesthetics. He names them the *Eliza Effect,* in which audience expectations allow a system to appear much more complex on its surface than is supported by its underlying structure; the *Tale-Spin Effect,* for works that fail to represent their internal system richness on their surfaces; and the *SimCity Effect,* pertaining to systems that enable the audience to build up an understanding of their internal structure.

Games designer Jason Rohrer, in his lecture at GDC 2009, *Beyond Single Player*, discusses what it takes for a game to be infinitely replay-able. In his analysis of *Spelunky* (Yu, 2009), he compares the aesthetics of the procedurally generated worlds of Derek Yu’s game to the fixed designs of Pac-Man and Tetris, saying: *“Chaotic-system reflex challenges and random sequences just don’t have the texture and atmosphere that explorable content has,”* and asks us, as designers: *“How do we craft textured, atmospheric single-player games that are still challenging, yet never tedious?”*

Spelunky provides a very satisfying answer to this question through its mixture of familiar and un-familiar game mechanics, surprising twists of random generation and constant presentation to the player of risk-reward decisions. A departure from the normal roguelike graphical forms and sheer variety of design and combinations make otherwise tedious or annoying challenges appear fresh, interesting and exciting to the player.

Joshua French, journalist for *1up.com* has analysed many of the RPG titles released in the last five years, in an effort to identify the aesthetics contributing to the appeal of the genre and while he does not strictly separate procedurally generated elements from any other aesthetic feature, he does start to produce an interesting taxonomy that will help to guide the analyses carried out in the next chapter. His article, *Return of the Dungeon Crawler* (2012) suggests many of the appealing elements contributing to the re-emergence of this kind of fantasy roleplaying game where generative design finds a natural home; some of the more useful tropes to analyse are: diverse dungeon-esque locales for exploration within an open-world setting (ideally suited to any procedurally generated level system), surprising and difficult combat encounters (explored in detail in the analysis of Final Fantasy in the next chapter 2.7), diverse character creation options (examined in the breakdown of The Sims, chapter 2.5) and strategies of progressive difficulty (see Desktop Dungeons analysis, chapter 2.8).

Randomised content has a lot of obvious appeal, from instilling value in replay-ability to providing infinitely interesting exploration and surprising, random encounters. A more subtle appeal is in line with Wardrip-Fruin’s *SimCity Effect* idea (2009, p. 15) and one of the fundamental elements which adds value to games within the roguelike genre (see chapter 3), which is that of the player building up an understanding of the internal structure of the system. This last aspect can be seen wherever a procedural system is founded on minimal rules-sets but provide massive variety through complex random connections (see Minecraft analysis, chapter 2.1 & High value factors in the Rogue: Berlin interpretation, chapter 3.3a). The strongest example appears to be the idea of randomly attributed elements such as magic items which have unknown properties selected from a wide range of options.

In summary it would seem that despite the radical progression of computing power beyond the need for data-based optimisations, many of the initial factors which instigated the emergence of procedural generation are still valid in today’s videogame titles. The most significant of these elements are highlighted and analysed in examples in the next chapter.

# 2: Generative Systems in Games

*“DungeonQuest is unfair!” – Minecraft splash screen.*

This chapter will begin to identify some implementations of procedural aesthetics in games which have been accepted well by critics. Each example demonstrates a particular element which is, in one way or another, fundamental to the success of its aesthetics and the following analyses discuss: the element’s position within the game’s mechanics and overall aesthetic, factors attributing to the success of the overall aesthetic and, where possible, mention of the methodologies of the element’s implementation.

Figure 1: Minecraft Landscape

1. 3D world generation in Minecraft (Mojang, 2009)

Before every new game of Minecraft, the game’s engine generates a vast and varied landscape using a set of carefully developed generative algorithms which ensure an even distribution of the game’s many resources while producing predictably designed aesthetic structures. Factors which help to scatter the resources are separations in the world’s ‘blocks’ such as underground and over-ground and the biome system (which regulates what resources are placed in which part of the world using ‘natural’ simulations). The algorithmic placement of each of these features also serves to provide support to the aesthetics of the overall landscape, for example rolling plains can be made to more frequently be generated near to hills, light woods or meadows as they do in nature.

**Figure 2: Spelunky - difficult challenges**

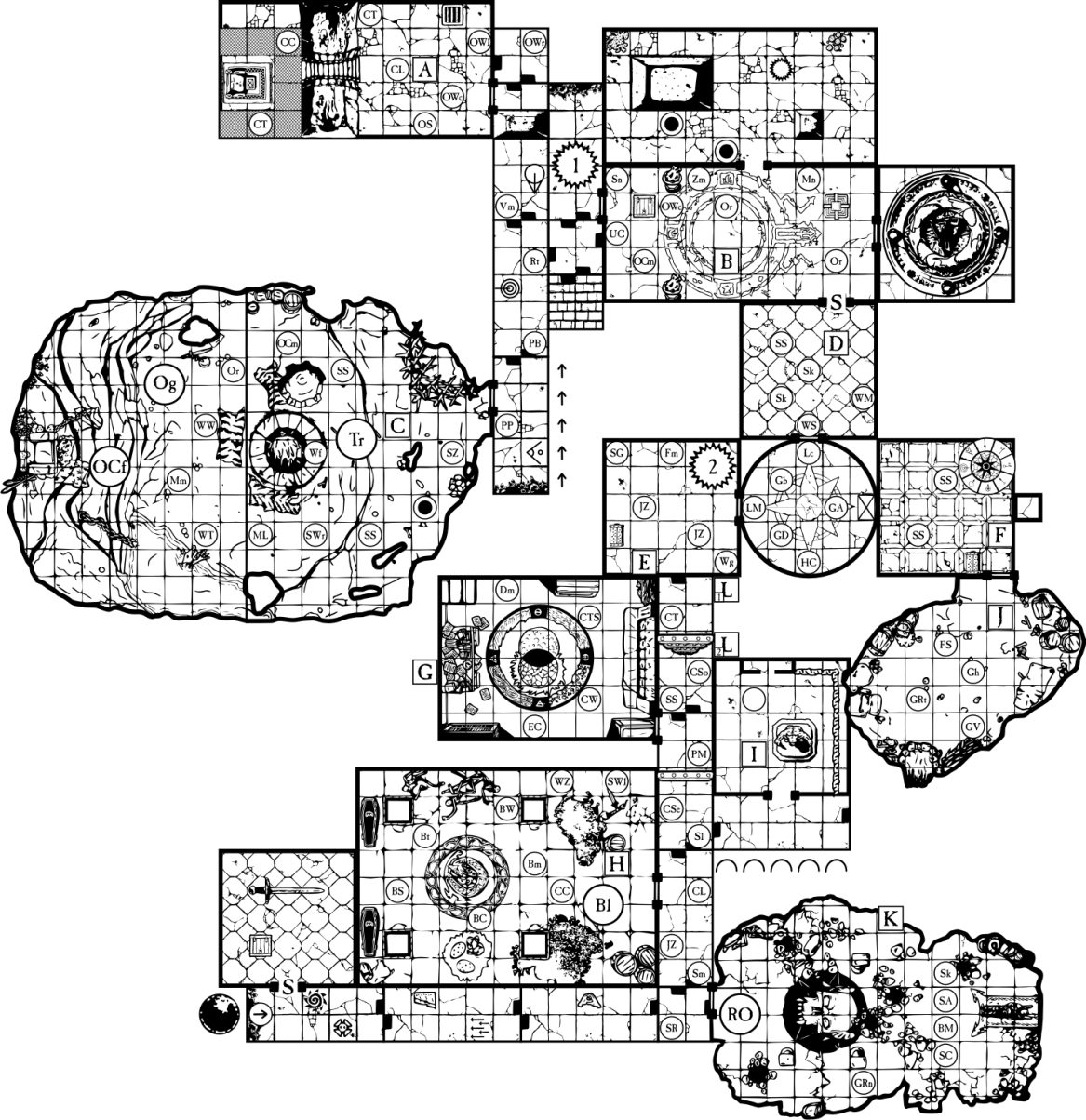
1. Interesting challenges in Spelunky (Yu, 2009)

As already specifically mentioned in the previous chapter, one of the noteworthy mechanics of Spelunky is its challenge which, as critics have pointed out, the game presents regularly but at random to the player. With the help of a tool created by enthusiast Darius Kazemi (2013) we can see in action the layout algorithms used by the game to generate its levels and from this exercise it is clear to see, as Kazemi concludes, that: *“by using an elegant combination of handmade layouts and probabilistic tiles, the game generates human-feeling challenges while keeping things fresh.”* This template system is a major key to creating good modular designs as described in chapter five.

**Figure 3: Advanced Heroquest - Modular board layout**

1. Board layout in Advanced Heroquest (Games Workshop, 1989)

Advanced Heroquest is designed with a modular playing board consisting of passage pieces with junctions, room pieces, doors and counters. The design has been modulated in such a way that dungeon environments can be easily and creatively designed by players, or constructed at random using dice rolled against a set of probability tables.

****

**Figure 4: Advanced Heroquest - Custom-designed level layout**

As there are very few examples of board games with a more complex system of environment generation, it can be fairly safe to assume, as discussed in the first chapter, that this level of generative layout approaches the point where a computer would be appropriate or even necessary to expand on its complexity. However complex the generation process is, the basic, structural level layout is then further complicated by furniture, hazard and monster pieces added depending on activity and further, tabulated dice-rolls.



**Figure 5: Fluxx card types**

1. Card-based rules system in Fluxx (Looney, 1996)

This entire card game is procedurally generated by the order of the deck and the choices of the players. The premise of the game is that players draw one card from the deck then play one card, aiming to obtain the necessary ‘keeper’ cards to match the ‘goal’ card which is in play. Many cards have rules elements printed on them which when played become assimilated into the game’s current rules set such as to, for instance, increase the number of cards drawn per turn or change the objective of the round.

**Figure 6: The Sims (3) - Choosing character behaviours**

1. Character design & behaviour in The Sims (Maxis, 2000-2013)

While there are a great many aspects of The Sims that are designed modularly for generative assembly, the most notable throughout the franchise is the design of both the characters’ forms and their behaviours. All of the iterations of these games have included a high level of character customisation features for the player and the algorithms alike, once more demonstrating the usefulness of a system which allows the player to design elements using the same pieces as the engine’s generative algorithms. An extra dimension in many Sims games is available to the player via the modding systems where new elements for characters can be added to the catalogue of parts and these additions contribute to the randomly created characters as much as they do to the options for the player. The behaviour controls for characters in the Sims games are dealt with using very simplistic options that again can be assigned at random as readily as chosen by the player; these patterns are basically ‘traits’ which dictate the behaviour of the AI controllers (for instance an ‘untidy’ character will pay less attention to mess in the environment than a ‘neat’ character).

**Figure 7: Cloudberry Kingdom - A procedurally generated 'hint-path'**

1. Procedural help/tutorial in Cloudberry Kingdom (Pwnee, 2013)

In much the same way as Spelunky generates levels with a solution built in, Cloudberry Kingdom, which delivers procedurally generated levels that are solved with timing-based, rhythmic movement, presents a guide solution in the form of a ‘ghost’ character that travels through the level at the same time as the player. Here we can see that a behind-the-scenes mechanic used by the engine to verify the solution for the generated level can also be offered to the player by way of a tutorial.

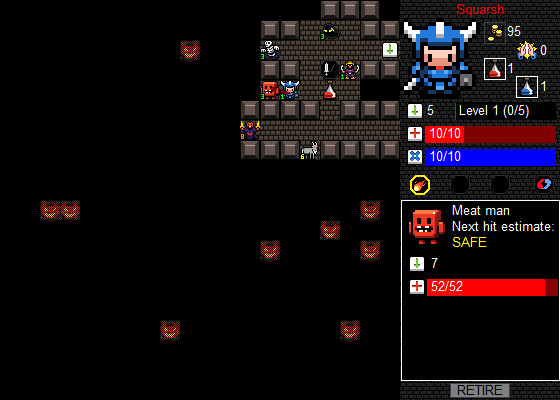
Figure 8: Final Fantasy (9) - Wandering in the wilderness



**Figure 9: Final Fantasy (9) - Random battle encounter**

1. Random encounters in Final Fantasy (Square, 1987-2013)

Most of the Final Fantasy titles include the random encounter element when the player character is travelling in wilderness areas. The element generally appears to be implemented on a footstep timer where every *nth* footstep taken by the player will yield a chance of an encounter. The type and strength of an encountered enemy is then usually chosen at random for options based on factors such as environment type and character level. Balance is often then mitigated by the game designers using limiting mechanics such as an item of equipment which increases or decreases the chance of an encounter occurring.

Figure 10: Desktop Dungeons - Optimal mob spawning for progression

1. Progression & difficulty in Desktop Dungeons (Joubert, 2010)

Desktop Dungeons has many generative elements: random maze-maps, item distribution and enemy placement, to mention a few. The game’s progression and difficulty systems are intrinsically linked to these other elements such that in order to enable the player to progress in terms of character level, appropriate entities (items & enemies) need to be generated at suitable distances from the character’s starting position.

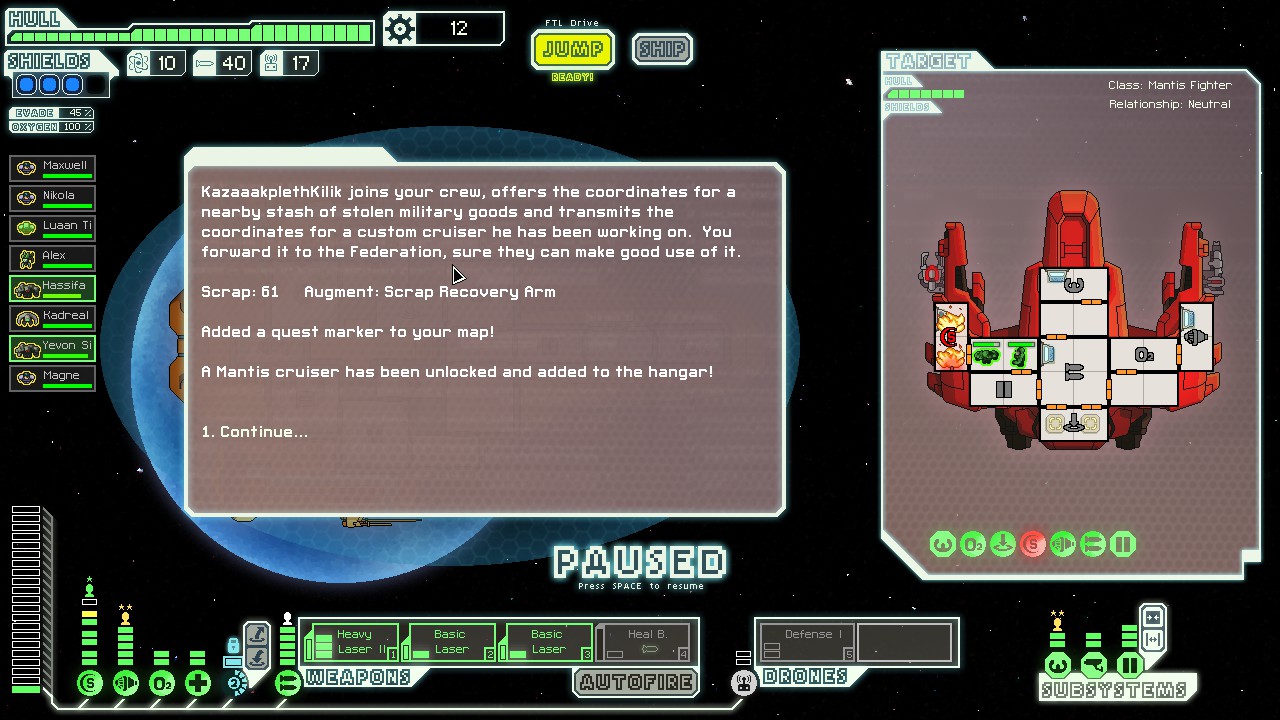
Figure 11: FTL - A quest event is encountered

Figure : FTL - Quest marker shown on star map

1. Random quests in FTL: Faster Than Light (Subset, 2013)

To break up the exploration of space in this ship-based roguelike, there is a chance that any encounter the player faces could initiate a quest. When this happens, generally a quest goal is established in one or more other locations on the current map with an appropriate event set to trigger when the player arrives. The pace of the game as a whole limits the length of these quests to two or three sub-quests at most and rarely continues a plot-line onto another map.

Figure 13: Proteus - Procedural world; entirely generative soundtrack

1. Generative music in Proteus (Curve, 2013)

Modular music is not a new thing in design terms; indeed many composers have been constructing modular music components for media such as film for many decades. Proteus handles modular music in a completely interactive way: every aspect of the interactions in the game trigger different sounds which merge to form a melodic piece. Aryn Clark, writing for *thephonograph.co.uk*, suggests in his article (2012) that the player triggering sounds through interaction follow the same principles of improvised music. Having been fortunate to attend a live screening of the game, with David Kanaga actually performing live, improvised music I find myself in complete agreement, being unable to easily distinguish the live experience from the procedural one.

#### Conclusions drawn from analyses

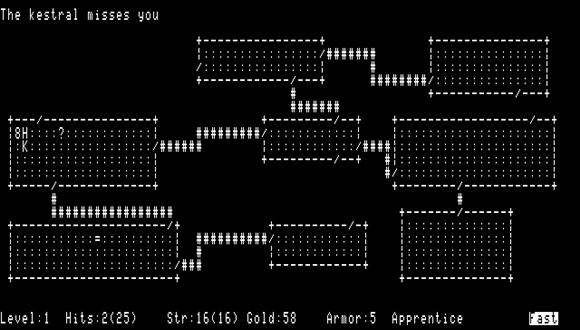
1. Game designers looking to include 3D world generation, whether or not including resource distribution in the algorithms, will find benefit in thinking about defining regions of particular terrain types. This method will improve the landscape’s natural aesthetics (provided that the algorithm simulates an observed, natural model) and give the designer more control over the behaviour of the algorithm.
2. While generating the playable world, 3D or otherwise, human-feeling challenges can be inserted at the whim of an algorithm. Combining basic puzzle, maze and encounter templates with interchangeable, generative pieces (e.g.: obstacles, enemies and hazards) will provide a large variety of interesting challenges to the player.
3. Keep in mind the processing power of the game’s generative algorithm; if the procedure for generating the playing environment involves players rolling dice, consulting probability tables and placing board tiles, then the level of complexity should be appropriate. Complexity can be added to basic structure in many ways independent or interconnected; furniture, hazards and monsters added at random arrangements (or indeed, applying wisdom from the previous analysis, using templates and random elements) can add many layers of depth to an otherwise simple environment.
4. It is not just the gameplay entities (such as items, furniture or enemies) that can be generative elements. Rules of play can be designed as modular components of an evolutionary rules-set and then generated with an algorithm.
5. Many of the games analysed here offer players the option to design elements using the same construction pieces as the procedural generator. This example of emergent affordance is very valuable to the game designer as it means that whenever a modularly-designed element works with a distribution algorithm, it can often also be presented to the player to increase interest in terms of customisation.
6. Sometimes aspects of a game’s behind-the-scenes algorithms, such as a level generator which needs to plot a successful route in order to verify that the course can be completed, can be re-packaged and delivered to the player for added value.
7. When generating encounters be mindful of not only the chance the event will occur, but also the type and level of the encounter versus the abilities of the player. A carefully constructed algorithm will scale these factors accordingly and giving the player skills or items which limit or extend these factors can be a good way to solve balancing issues.
8. A sophisticated algorithm will present opportunities to the player that can be taken at the character’s current level. Working progression mechanics into a system such as this can be done by occasionally offering extremely challenging opportunities, gauged on the character’s current level (or the level they reasonably could be expected to have accomplished at that stage).
9. The random quest aspect speaks back to the interesting challenge dynamic but in this context can be used to deliver a narrative aesthetic. Here it would pay to explore the modular aspect of a quest generator and have many possible outcomes to each decision or task which can be designed using a template system similar to that previously discussed.
10. Whether the algorithm generating music for a game is simply triggering complete loops of music or effectively constructing a symphony of generative music from an orchestra of interactive game entities, modular music has and will always be an important aesthetic device for most games’ design.

# 3: A Rogue Gallery

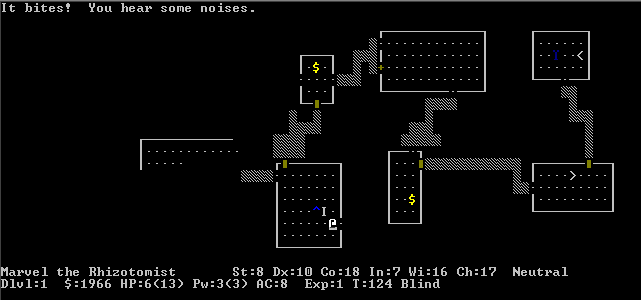
*“Welcome to the Dungeons of Doom.” – Rogue intro text.*

This chapter provides an analysis of the Rogue paradigm in history and modern games plus a discussion of the popularity of such games today, referring to the timeline offered by Joshua French (2012) and his opinion about what makes these kinds of games even more popular with modern players.

Figure 14: Rogue as it looked on a UNIX terminal

1. The original game

The classic videogame *Rogue* (Toy & Wichman, 1980) epitomises the appeal of procedurally generated game content. This incredibly simple Unix-developed RPG was designed by two students on a university network and for the major part of its early life it was also played on these kinds of networks. At the time of Rogue’s inception arcades were populated by interesting graphical games the likes of Space Invaders and Pac-Man while the university computers had nothing to offer other than basic word games and imagination-provoking text-based adventure games. This contrast led the creators, Michael Toy and Glen Wichman, to consider the aesthetic possibilities for their favourite genre, the adventure game, researching and developing a method to deliver what was to essentially be a graphical version of a role-playing adventure game on their Unix timesharing systems. Knowing the systems’ limits and leveraging newly-developed cursor-placement technology emerging from their university, Toy and Wichman set about designing a graphical system that could be displayed using only ASCII symbols, with pipes (|) and hyphens for walls and slashes for doors, coloured letters for various entities such as monsters and loot items and the @ symbol to represent the player’s position. Player controls were updated from typing the directions north, east, etc. to using the keyboard’s cursor arrows and the verbose descriptions of the text-based adventures were reduced to single-line statements describing the entities encountered by the player during movement. The developing duo quickly realised that storing pre-designed levels in the 128kb data-space provided by their then powerful computers was unfeasibly limiting and so devised an innovative engine that would generate level layouts and entity placements at random. This gave way to a powerful emergent idea in the field of games design, summed up very well by the words of Toy himself: “The sad discovery for authors of text-style adventures is that it is not that fun to play your own game. You already know all the solutions to the puzzles. The greatest part of Rogue, and the part I still wish for as I look at the gaming scene today, is that it made a new world every time.” (Edge, 2009). With this quote the mechanic of surprise for the author and player alike presents itself as one of the chief aesthetic considerations in generative games design. The other outstanding mechanic which belongs in this game’s aesthetic set is the ramping difficulty; as the player descended toward the lowest level of the random dungeon in order to retrieve the fabled *Amulet of Yendor* the challenge of the game became more and more difficult. Here this idea is firmly set into the aesthetic goal of emulating the pen-and-paper game *Dungeons & Dragons* (Gygax & Arneson, 1972) by presenting this difficulty transition through a vague narrative of a capricious, power-mad Dungeon Master. The difficulty was made even more pertinent by the concept of permanent death: on dying the player had to restart the entire game, playing now with a completely new character in a totally new dungeon.

Figure 15: NetHack's visuals did not depart far from Rogue

1. Early variations

Following the success of the first few versions of Rogue, many other platforms (C64, IBM/PC, Atari ST, etc.) would receive iterative ports and adaptations, still under the original franchise. As these versions were developed the core aesthetics were largely untouched with the only improvements to the style coming in the form of increased graphic and audio fidelity and this was the secret to the game’s continuing success. At the same time as the original game was being catholicised across platforms, new games inspired by the simple framework of Rogue had started to emerge, the most outstandingly successful example even to this day being the almost identically framed game *Hack* (1984) which was closely followed by the highly iterative *NetHack* (1987) which as a franchise has spread across many more platforms and versions than the original Rogue could have ever hoped, while the core game NetHack has only recently moved into its version 3.4.3 (2010). NetHack like its spiritual ancestor has stayed very true to the core aesthetics whilst simply improving fidelity amongst its elements (graphics, sound, variety, etc.). Many types of games were designed based on the format of Rogue following its success on the university networks – some much more ambitious and diverse than others. Significant titles included *Ancient Domains of Mystery* (Biskup, 1994), *Linley’s Dungeon Crawl* (Henzell, 1997) and *Angband* (Cutler & Astrand, 1990) which augmented the original aesthetics with narrative taken from the writings of J. R. R. Tolkien. This was followed by a spate of Japanese titles under the franchise *Mystery Dungeon* (‘Fushigi no Dungeon’ in Japanese) which started with the Super Famicom title *Torneko’s Great Adventure* (Chunsoft, 1993) where the graphics received a massive, 16-bit overhaul and brought a vibrant, zany feel to the aesthetic setting. Other games which have been inspired to pick mechanics from Rogue for their own aesthetic include *Diablo II* (Blizzard, 2000), which took the permadeath feature and the incredibly complex *Dwarf Fortress* (Adams, 2002) which took on nearly all of the original aesthetic set and added many mechanics and graphical diversifications.

1. The Berlin Interpretation

The popular uptake of the Rogue format has given rise to a great number of imitations between then and now, known under the genre title *Roguelike*, so much so that in 2008 an annual international conference for development was established where conventions were laid out as guidelines for creators of games in the genre. These conventions which are not intended to restrict progress but instead to help to describe the aesthetics of the genre, are divided into two sections: high and low value factors.

1. **High value factors**
2. Random environment generation

The game world is randomly generated in a way that increases replay-ability. Appearance and placement of items is random. Appearance of monsters is fixed, their placement is random. Fixed content (plots or puzzles or vaults) removes randomness.

1. Permadeath

You are not expected to win the game with your first character. You start over from the first level when you die. (It is possible to save games but the save-file is deleted upon loading.) The random environment makes this enjoyable rather than punishing.

1. Turn-based

Each command corresponds to a single action/movement. The game is not sensitive to time; you can take your time to choose your action.

1. Grid-based

The world is represented by a uniform grid of tiles. Monsters (and the player) take up one tile, regardless of size.

1. Non-modal

Movement, battle and other actions take place in the same mode. Every action should be available at any point of the game. Violations to this are ADOM’s over-world or Angband’s or Crawl’s shops.

1. Complexity

The game has enough complexity to allow several solutions to common goals. This is obtained by providing enough item/monster and item/item interactions and is strongly connected to having just one mode.

1. Resource management

You have to manage your limited resources (e.g. food, healing potions) and find uses for the resources you receive.

1. Hack ‘n’ slash

Even though there can be much more to the game, killing lots of monsters is a very important part of a roguelike. The game is player-vs-world: there are no monster/monster relations (like enmities, or diplomacy).

1. Exploration and discovery

The game requires careful exploration of the dungeon levels and discovery of the usage of unidentified items. This has to be done anew every time the player starts a new game.

1. **Low value factors**
2. Single player character

The player controls a single character. The game is player-centric, the world is viewed through that one character and that character’s death is the end of the game.

1. Monsters are similar to players

Rules that apply to the player apply to monsters as well. They have inventories, equipment, use items, cast spells, etc.

1. Tactical challenge

You have to learn about the tactics before you can make any significant progress. This process repeats itself, i.e. early game knowledge is not enough to beat the late game. (Due to random environments and permanent death, roguelikes are challenging to new players.) The game’s focus is on providing tactical challenges (as opposed to strategically working on the big picture, or solving puzzles).

1. ASCII display

The traditional display for roguelikes is to represent the tiled world by ASCII characters.

1. Dungeons

Roguelikes contain dungeons, such as levels composed of rooms and corridors.

1. Numbers

The numbers used to describe the character (hit points, attributes, etc.) are deliberately shown.

Here the separation of high and low value factors can be leveraged by games designers as these categorisations represent popular appeal: the high value factors are the aesthetic elements which make roguelike games appealing to a player and a designer using them as a toolset for designing generative games would be designing a game with engaging aesthetics; the low value factors are still of interest to players but would tend toward those players already deeply invested in the genre.

1. Modern emergence

It is clear to see that many RPG games have drawn influences from the aesthetic and technical design of the original Rogue and in recent years the number of games being developed in the roguelike genre has increased at great speed. The emergence of this genre and increase in public awareness could be attributed to the various homages to Rogue contained in recent RPGs such as *Chocobo’s Mysterious Dungeon* (Square, 1997), spawned out of the *Final Fantasy* franchise and the *Shining Soul* (Nextech, 2002) roguelike offshoot of Sega’s *Shining Force* games.

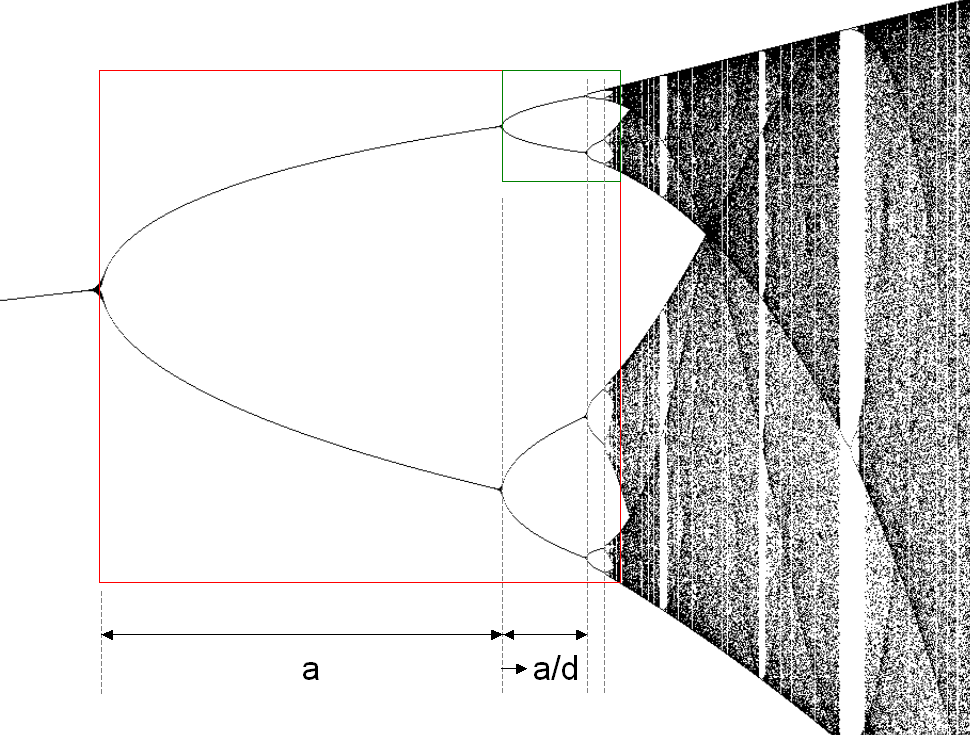
In the last few years there has also been a rise in the number of independent games within the genre, ranging from direct iterations of the original such as *Dungeons of Dredmor* (Gaslamp, 2011) which follows all of the high value factors of the Berlin interpretation and adds its own visual and humorous aesthetic qualities, to the critically acclaimed title *FTL: Faster Than Light* (Subset, 2012) which provides procedural quests for challenge and intrigue. Many of these indie titles have sought to improve on the core aesthetics, mainly through graphics or mechanics. A particularly well thought-out example being *Rogue Legacy* (Cellar Door, 2013) which innovates around the permadeath mechanic, adding an inventive progression system where characters level up and change every time the player dies.

# 4: Mathematical Aesthetics

*“Human was the music, natural was the static.” – John Updike*

Since a little before the advent of home computing in the 1970s enthusiasts, designers and theoretic mathematicians alike have experimented with many naturally occurring mathematical systems, using them to explore various aesthetic constructs such as fractal imagery, cellular patterns and procedurally generated music. This chapter investigates the development of these systems and discusses ways in which games designers can leverage their functions for generative elements.

Figure 16: Bifurcation diagram

1. Chaos

Mathematician James Yorke was the first theorist to delve into the meteorological works of Edward Lorenz (who devised the well-known concept of the ‘Butterfly Effect’) and experiment with these patterns and algorithms on a computer. Yorke coined the term *chaos theory* and during the 1970s with his friend, a biologist named Robert May, started to investigate many pattern-based phenomena using very powerful computers to visualise the effects of doubling simple equations. Their research would quickly lead to the discovery of fractal imagery but not before the pair of researchers had explored massive amounts of behavioural patterns. For games designers the importance of their enquiry lies in the ‘bifurcation diagrams’ (see fig. 16) that May and Yorke produced in their conclusions, where the results of each extensive exploration were assembled into a single picture showing how changes in one parameter would change the ultimate behaviour of the simple system (Gleick, 1988, p. 71). While this type of function is not immediately useful to every games designer, the probabilistic patterns can be advantageous to an AI programmer trying to govern complex behaviours with simple parameters or, as demonstrated in a very short-lived MMO, *N.E.O Online* (Sonov, 2009), for designing a world in which small changes made by players could end up impacting on the world in a large-scale way.

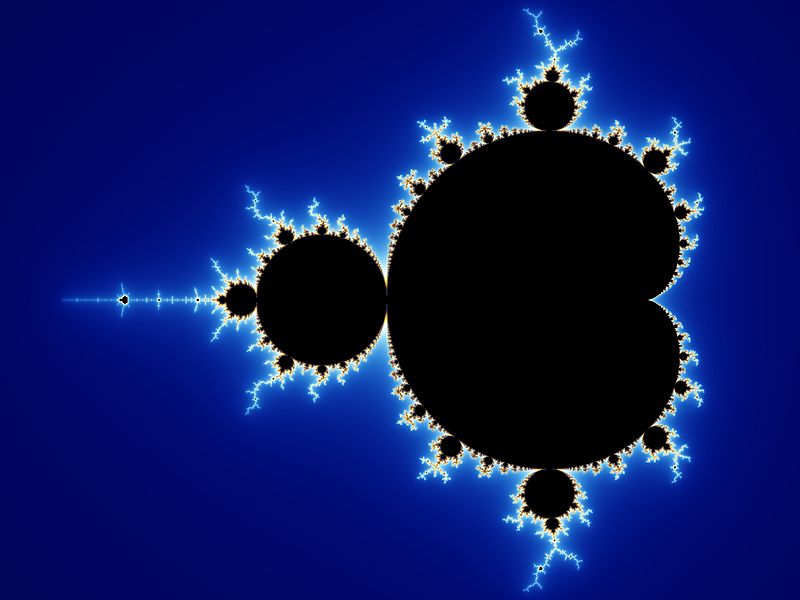


Figure 17: Mandelbrot figure

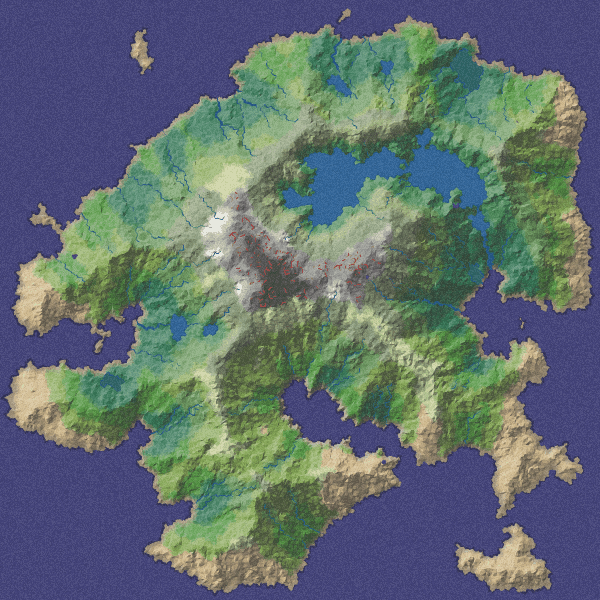
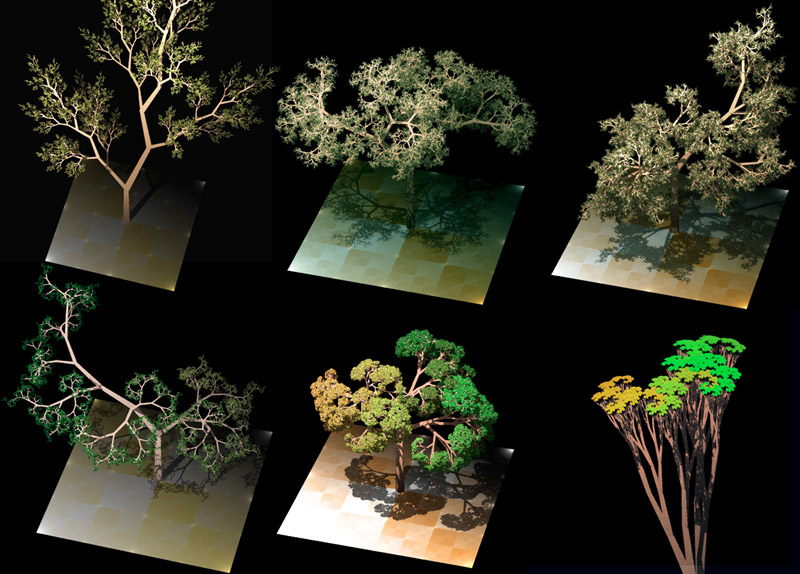
****

Figure 18: Coastline data generated using fractal algorithms

1. Fractal sets

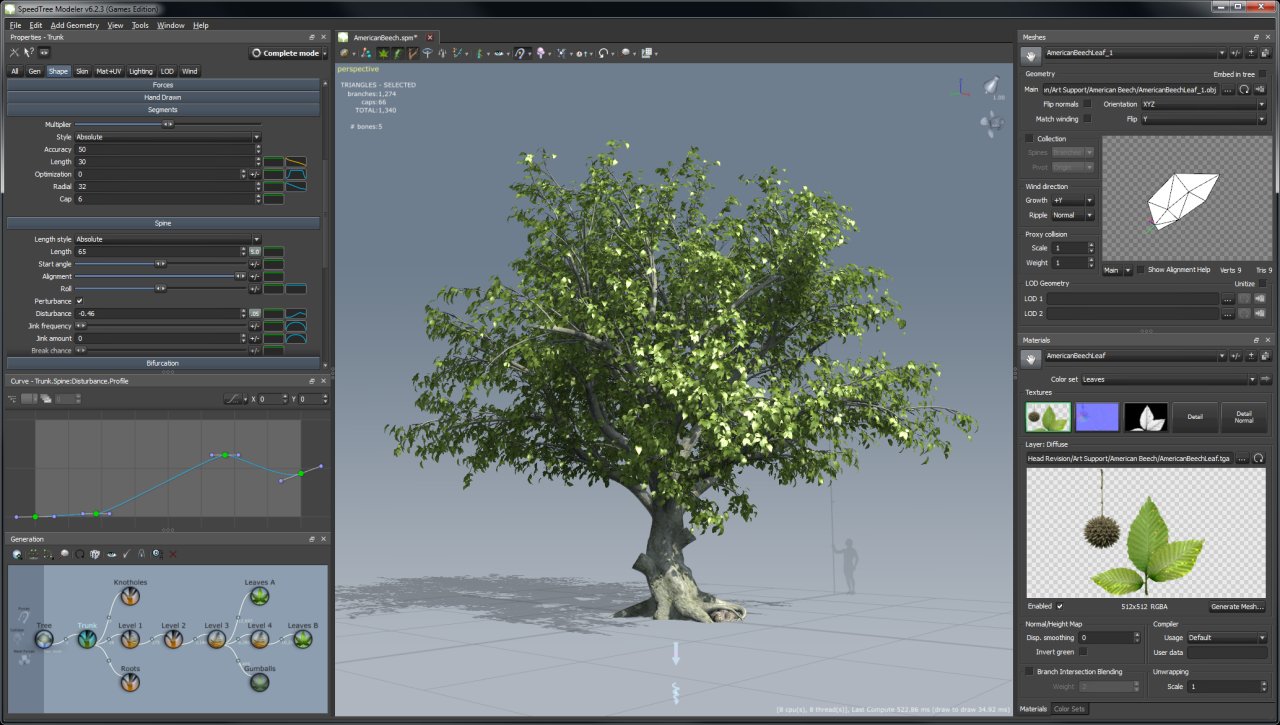
Fractal patterns exemplify some of the most recognisable mathematical imagery that was being explored in the microcomputer age and many enthusiasts will be familiar with the term ‘Mandelbrot set’, or at least will recognise the classic, heart-shaped Mandelbrot figure (see fig. 17). Benoit Mandelbrot’s most famous academic research started with his ‘theory of roughness’ which, in simplistic terms, concluded that naturally occurring coastlines (and indeed any other natural form which exhibits ‘rough’ edges) can be thought of as having effectively infinite length – depending on the length of the measuring device used. The theory is a thought exercise which follows as such (Gleick, 1988, p. 94): if a surveyor were to set his dividers to a length of one yard and walk them along a coastline, the resulting number of yards would be an approximation of the coast’s length because the dividers skip over many twists and turns smaller than one yard; if the surveyor were then to reduce the length of the dividers to one foot and take a new measurement he would arrive at a significantly greater length because the dividers would now capture more detail. Each reduction to the metrics would subsequently result in greater lengths of measurement and it was at the conclusion of this thought exercise that Mandelbrot discovered the recognisable patterns associated with his name. Today almost all software used by games designers and digital artists to produce natural, rough-edged forms makes extensive use of the fractal mathematics devised by Mandelbrot (see fig. 18).

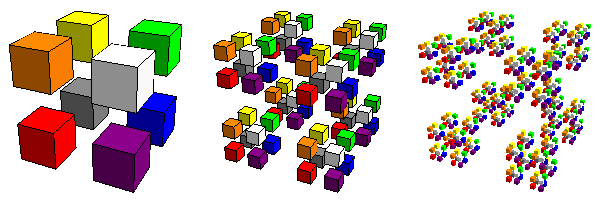
Figure 19: Plant forms generated using l-systems

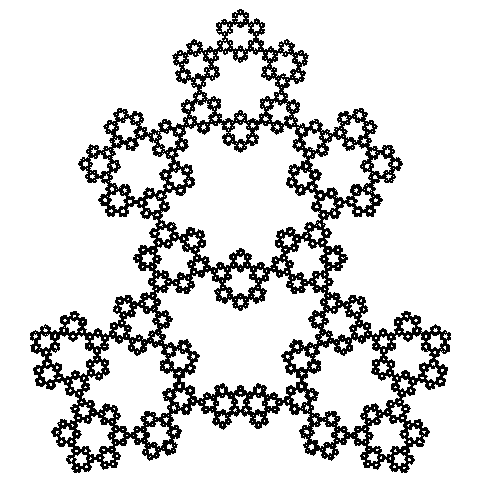
****Figure 20: 3D tree designs formed from l-systems

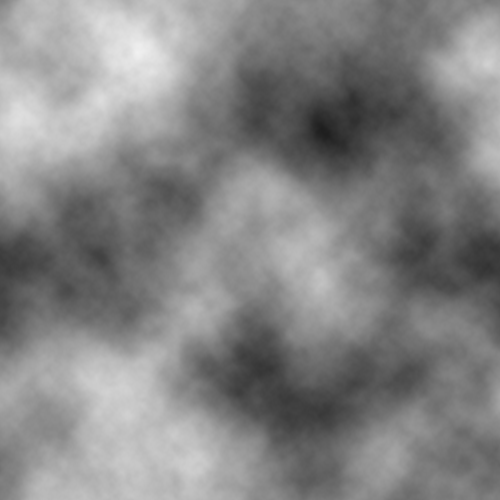
1. L-systems

Closely related to Mandelbrot’s theory of roughness is a series of algorithms developed by theoretical biologist, Aristid Lindenmeyer, used as a set of production rules to describe and generate a simulation of the natural growth of plants. Many plant generation programs (such as *SpeedTree*, IDV, 2009) use these functions to produce very detailed and realistic trees and plants. Lindenmeyer’s sets combine theories from many mathematicians working in the field of fractals and other cellular simulations of infinite forms such as Cantor (see fig. 22) and Koch (see fig. 23) and can thus be described themselves as fractal forms.

Figure 21: SpeedTree software used to generate tree assets for games

Figure 22: 3D iterations of Cantor's Dust algorithm

Figure 23: An arrangement of Koch Snowflakes

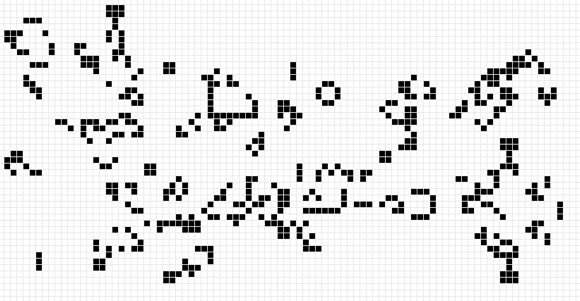
****

**Figure 24: Basic Perlin Noise**

**Figure 25: Reticular simplex noise forming clouds**

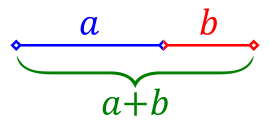
1. Perlin noise

One of the most useful and innovative mathematical functions used by graphics programmers is the Perlin noise algorithm, developed by Ken Perlin, professor of computer science at New York University. The algorithm generates an infinite field of pseudo-random noise, with every aspect of visual detail set at the same size, making it readily controllable and thus ideal for procedural textures. Iterating on his own function, Perlin went on to develop *simplex noise* which is a more efficient form of the same algorithm that also lends itself (due to lower computational overheads) to reticular, fractal patterns.

Figure 26: John Conway's Game of Life

1. Cellular automata

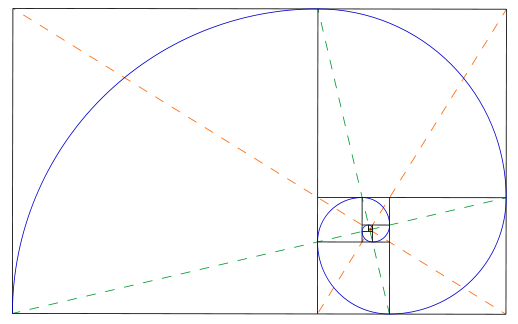
John Horton Conway, a mathematician working in the fields of number theory and recreational mathematics, devised a cellular automaton known as *Conway’s Game of Life* (Conway et al, 1983, p. 817-850). Like other examples of this kind of mathematical model, the *Game of Life* simulates the life cycle of binary, abstract cells and as well as being a stand-alone, experimental, zero-player game, it can be leveraged by games designers (AI developers in particular) to describe many conceptual patterns of growth within procedural systems such as population controls or spreading distribution patterns.



**Figure 27: A line depicting the golden ratio**

1. Golden ratio

The golden ratio, referred to in mathematics with the Greek letter phi (*ϕ*), is well known to many artists having been used for centuries as a guide to composition and anatomical measurement (indeed the letter phi was the first letter of the name of the Greek sculptor, Phidias). Phi’s usage has become common place because of its ubiquitous appearance in nature and so finds its way into everything from geometric calculations to aesthetic constructions. The maths can be understood in very simplistic terms as roughly the ratio one-third to two-thirds of a measure and has an equally basic algebraic definition such that: in a line divided into two segments, *a* and *b*, *a* + *b* is to *a* as *a* is to *b* (see fig. 27). It has been known and commented upon for a very long time (Hemenway, 2005, p. 20-21) that most aesthetic forms derived from the golden ratio will be *“naturally pleasing”*. Additionally of interest to the following section, Leonardo of Pisa (known as Fibonacci) mentioned in his *Liber Abaci* (Sigler, 2002) that *“the ratio of sequential elements of the Fibonacci sequence approaches the golden ratio asymptotically”.*



**Figure 28: The golden ratio and the Fibonacci**

**sequence are both found in this spiral pattern**

**Figure 29: The same spirals are also found in**

**nature, as seen here in the leaves of an aloe plant**

1. Fibonacci sequence

As already mentioned in the previous section, the Fibonacci sequence is highly interdependent with the golden ratio – many forms drawn using the golden ratio will exhibit sequences within Fibonacci and vice-versa. The factor which initially makes these number sequences stand out from any other generated around the golden ratio is that due to their unending nature, they can be quite useful in the generation of seeded, pseudo-random sequences, in particular the algorithm devised by David Braben while developing the star fields of Elite.

1. Maze and path-finding algorithms

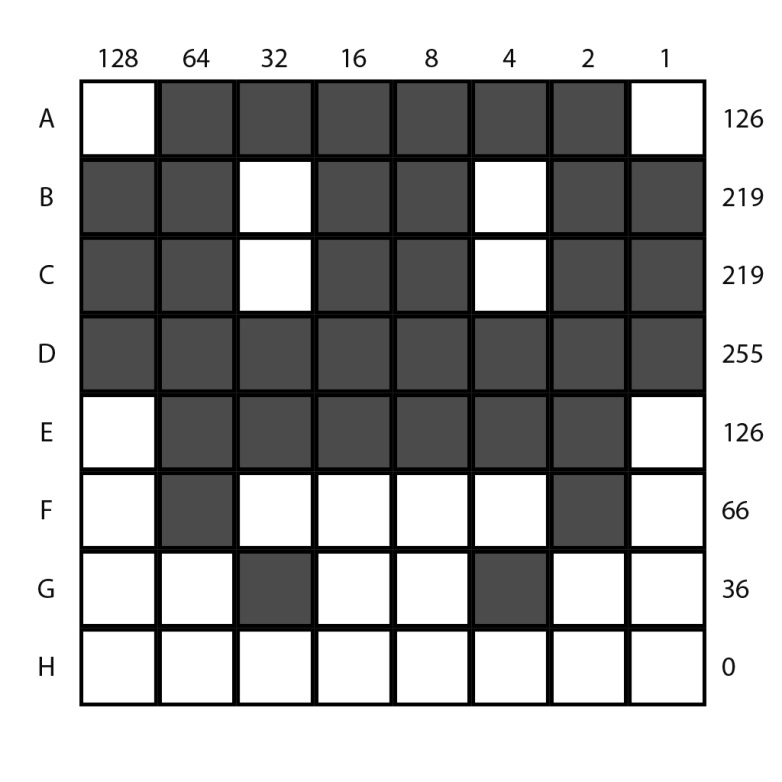
There have been many hundreds of algorithms devised within the mathematical fields of maze and path theory. Maze algorithms are generally constructed by first dividing the playing area into a grid before selecting a start point and an end point and then ‘walking’ through a randomized path to get from one end to the other; in this way a single algorithm can provide many variations of maze patterns and each time guarantee a fixed number of solutions. Path-finding in its simplest form can often easily be implemented using derivatives of the ‘A\*’ algorithm which functions such that all the possible nodes around a point on a grid are scored depending on their distance from the start point and then the end point, with the more expensive nodes being eliminated leaving an array of parented nodes which make a path. The intrinsic details of the various forms of these algorithms is beyond the scope of this paper to discuss and so for more information, many of the well-established patterns for both maze generation and path-finding are explained in *AI for Game Developers* (Bourg & Seemann, 2004).

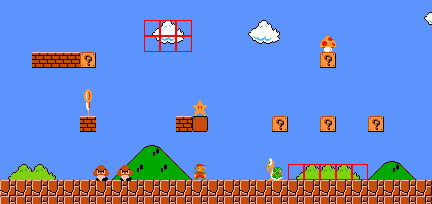
# 5: Modular Patterns

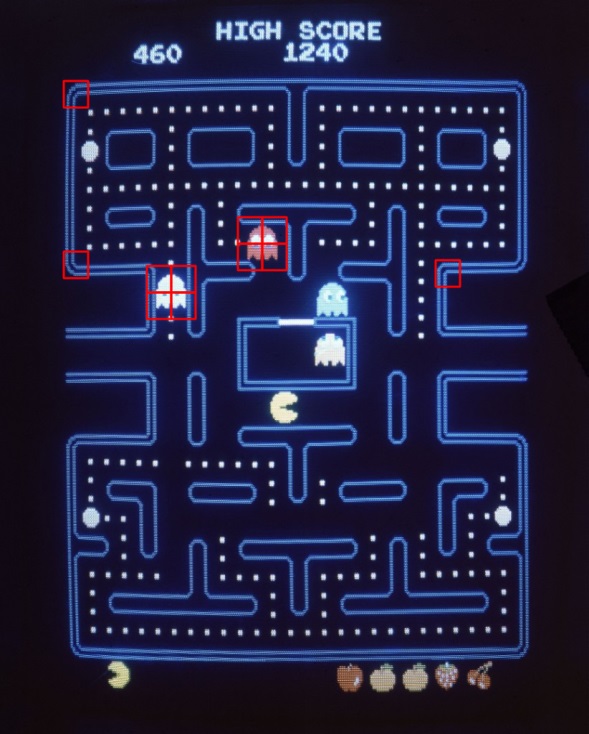
*“The art of simplicity is a puzzle of complexity.” – Douglas Horton*

Unless the game is founded on the principle of generating everything from pixels, voxels or points, then everything designed with a modular purpose will require good templates. When talking about art assets, especially modular environments, the term ‘prefab’ arises very often and many existing game engines (Unity & UDK are perfect examples) build these entities into their resource organisations. Prefabs are simply pieces of reusable design (generally geometry) which can be assembled by a designer or equally by an algorithm to create a larger, more complex structure and they are the smallest units a modular asset designer needs to consider. Prefabs also have the benefit of being commonly implemented as *instances* which means that changing one artefact will automatically propagate the change to every other example of that prefab used in the game; this can be a huge timesaver on a large-scale project.



**Figure 30: A bit-grid demonstrates how a moveable graphic can be encoded into a simple list of numbers**

Modular asset design can be shown to have arisen in the early days of 8-bit games when in order to produce graphics, a sprite needed to be designed and encoded onto a bit-grid (see fig. 30). The level of work required for each graphic was high and so workflows which reused assets were quickly adopted. As can be seen in a breakdown of many 8-bit games (see fig. 31 & 32) the elements designed to be reused were sparse with their detail so that they could be copied many times on the same screen without noticeable repetition. The same elements were often designed to be versatile enough that with a simple re-colouring they could be used to represent other aspects (see fig. 31).

****

**Figure 31 (above): Super Mario**

**Bros. (Miyamoto, 1985)**

**Looking at the red boxes, the**

**graphic for the cloud has been**

**simply re-coloured to be used for**

**the bush as well**

**Figure 32 (left): This can also be**

**seen in Pac-Man (Iwantani, 1980).**

**The red boxes on the corners of**

**the maze are mirrored and**

**repeated in other parts of the**

**level. The ghosts are also re-used,**

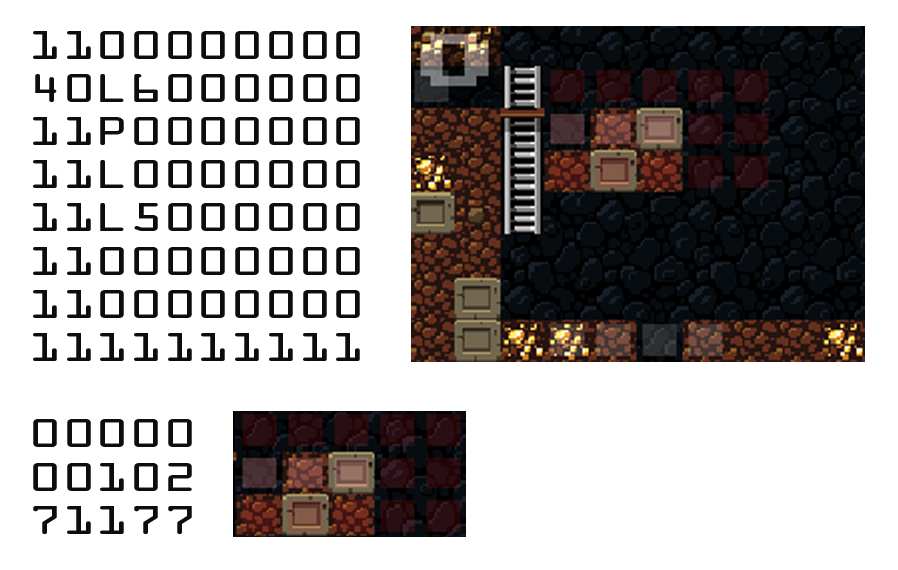
**while re-colouring them**

In his article published in an issue of *Game Developer Magazine* (2002, p. 30-35) level designer for Epic Games, Lee Perry recommends some important considerations when designing modular components and describes each aspect quite thoroughly. It is worth a note to say that Perry mentions the technological imperative of memory here (p. 35) thus demonstrating the return of the initial motivator identified in Braben & Bell (see chapter 1, p. 9).

If we take a look at a couple of the analysed examples from chapter two we can see how the procedural device is supported by the aesthetic, modularly designed assets.

1. Spelunky template example

The creator of the tool used in the analysis in chapter two, Darius Kazemi, was the programmer responsible for officially porting the game to HTML5 and so has some insight into the game’s templates which he kindly shares in the essay beneath the tool (Kazemi, 2013). From his description of a level space as a two-dimensional grid and using a simple encoding system to plot what he refers to as ‘*static tiles’* and *‘probabilistic tiles’* we can easily see how this translates to a tiled sprite-based representation by the game engine (see fig. 33). Just like in the Super Mario Bros. screen-capture above (see fig. 31) the game level is constructed from a set of repeatable, pixel-graphic tiles which in the Game Maker (YoYo,1999) engine can be assigned programmatic meaning through variables and code. The sprite graphic, along with its parameters and functions comprise the complete modular asset and can be referred to in the engine as a prefab, as discussed at the beginning of this chapter.

Figure 33: The code on the left is translated by the game engine into placement information for graphic tiles. The bottom inset shows a sub-template which is generated before being placed into the level

1. Modular assets in The Sims

The Sims franchise has always been primarily focussed on customisable characters and most titles have modding systems to enable third-party content to be added into the options. This is one of perhaps the widest implementations of customisability and so the original asset designers and programmers devising the systems have clearly paid close attention to the concept of reusability. For simplicity’s sake, take a look at the system implemented in the first title (see fig. 34) here characters are broken simply into head and torso and have a mesh and a texture map for each part. There are duplications for a fat, thin and fit character in each gender, age and each of three skin-tone variants (see fig. 35) and then each of those bodies is substituted out for a mesh which represents clothes. The clothes meshes are fewer than the texture maps for clothes not only due to colour changes but also because quite a lot of style can be differentiated with a well-designed texture map (see fig. 36).

Figure 34: The Sims (original) - demonstrates a simple modular character customisation tool



Figure 35: The Sims (4).

3 body types, each with

various textures for skin

tones



**a**

**c**

**b**

Figure 36 (a, b & c): The Sims (2) – the models in each image use the same geometry but have a different texture mapped onto each, making smart use of alpha maps to give the appearance of transformed geometry

An object-oriented programming language, due to its fundamental usage of data models is ideally partnered with a modular designed game system. These kinds of languages (such as C++ or Java) require the programmer to think of many aspects of the game and its data structures as component objects which assemble into more complex objects, so attaching this type of code onto a modular asset system is a very natural process. Some specific suggestions of how these two systems are natural simulacra are explained in *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma et al., 1994, p. 104-108).

From a careful analysis of a great number of varied modular asset sets, and from my own experimentation with sets I can deduce that the ethos of good modular design equates to something like: when trying to break items down to the ‘smallest’ component parts, aim for that fuzzy logic band between too big and too small. Lee Perry’s considerations begin with *scale* (2002, p. 32) where he directly deals with the issue of ‘how small should components be?’ suggesting settling upon a scale proportional to the scope of the game’s environment: *“If you’re going to be shoving a player through entire cities in a Porsche, you’ll want to break everything down into chunks of geometry as large as entire city blocks. If you’re going to be creeping a single player through the narrow, abandoned corridors of a derelict spacecraft, you’ll want to work at a very fine detail level with layers of components interacting in a complex fashion.”*

# Conclusions

This investigation could go on indefinitely as new games continue to innovate in a field of game design which seems to be becoming equally more interesting to both developer and player so this can only be a summary and is in no way conclusive. I have discussed, in regards to both old and new games, some of the motivations behind using procedural systems, identified and addressed a number of successful aesthetic devices which use generative design, pointed out some of the more known mathematical functions that can be easily further investigated by an interested reader and discussed a few methodologies for implementing some of the design elements highlighted by the research.

There appear to be two major motivations behind choosing to use a generative system in both early and modern games: the desire to surprise both designer and player, and the need to compress large amounts of data into a small space. The former of the two motivations is at least as important to today’s designers as it was in the 1980s, while the latter has become less relevant as technology advances forward. It is however fair to say when we see an emergent technology, such as Flash or mobile gaming, that the first wave of productions can tend toward this attitude before the new technology grows in its capacities. Additionally, in terms of motivations, the rise in interest in the roguelike genre must go some way toward proving a fan-based incentive behind developing these generative games, whether it is due to the fan-players’ demands or the passions of a nostalgic developer.

The aesthetic devices identified by this research will hopefully begin to establish a toolset that can be used for generative games design. Each of these elements has had only a small amount of exposure and criticism here but I believe them to be strong foundations for a useful toolset. For the sake of clarity, the ten aspects addressed in chapter two can be rationalised to the following considerations:

1. World generation, incorporating resource distribution and aspects of environmental challenge. Many of these elements can make use of a template system for modular design. Studying real-world topographical features can provide great benefit to a designer constructing elements for a believable world-map generator and classification systems useful to both aesthetics and mechanics can result from implementing a region-based distribution function.

ii) Modular character design and artificial intelligence. Both of these features have their own challenges even without the added complication of modularity but designers who can afford to develop these aspects may benefit from being able to offer extra customisability to the player both inside the game and externally within a modding system.

iii) Procedural emergence. Opportunities can be found to provide the player with a function which is part of the algorithms’ behind-the-scenes activities, such as using path-verifying functions as tutorials. Another opportunity for procedural emergence could also come from the template systems frequently referred to in this exploration, where the layouts are already input into the game engine and could easily be presented, in whole or in part, to the player by way of a map or guide to a level, or simply a catalogue of hazards or obstacles.

iv) Random encounters and progression. When designing these aspects it is important for balance to be able to read or predict the character’s current level or state. Armed with the correct information a very strong progression mechanic can be built into complex distribution algorithms, enabling human-feeling level designs to arise from generative layout systems.

v) Modular music is already a major part of a good game designer’s toolset, here we simply ask the question: how modular should the music be? The answer is of course different for every project, a horror game could exploit the repetitiveness and suspense of less loops while a more arty experience can afford to get involved with pure generative music, having every interactive aspect of .the game provide a cumulative musical cue.

vi) Modular design does not have to be limited to art assets and sound. Indeed many of the abstract elements of a game such as rules or quests can be produced generatively. As discussed in chapter five, modularity of design is heavily dependent in the first instance on the scale of the components the designer choses to create, but any additional planning time proportioned to this stage of the workflow will doubtless provide time-savings and natural problem-solving benefits later on in production.

A game designer looking to create a popular, procedurally generated game would be wise to pay attention to the findings of the exploration of the roguelike paradigm. The list of high and low value factors are very likely to provide deep insight into the desires and needs of players of these kinds of games and the sheer quantity of innovative games being produced in this genre should be a good indication that the framework is not too rigid to allow for wild experimentation with each and every one of these features.

#### Afterword

Procedurally generative systems have long been a great passion of mine in the fields of both mathematics and technical games design, and I am very pleased as we look down the future of games development that there seems to be no abating the interest and demand for such games from players and developers alike. It is my fond hope that the ideas I have proposed here will provide the foundations for a framework by which we can improve our approach to this kind of design and I fully intend to continue this formal exploration, building up a strong toolset of technical and aesthetic devices that will be useful to designers and developers of generative games.

# References

BOGOST, I. 2007. *Persuasive Games: The Expressive Power of Videogames.* Cambridge, MA: MIT Press.

BOURG, D. M. & SEEMANN, G. 2004. *AI for Game Developers.* Sebastopol, CA: O’Reilly Media, Inc.

CONWAY, J. H., BERLEKAMP, E. R. & GUY, R. K. 1982. *Winning Ways for your Mathematical Plays Vol. 2.* London: Academic Press.

EDGE STAFF. 2009. *Edge Online. The Making Of: Rogue, page 2.* [WWW]. http://www.edge-online.com/features/making-rogue/2/. (16 November 2013).

FRENCH, J. 2012. *Return of the Dungeon Crawler.* [weblog post]. *1up.com Video Game Reviews.* http://www.1up.com/do/blogEntry?bId=9107310. (21 November 2013).

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston, MA: Addison-Wesley.

GLEICK, J. 1987. *Chaos*. New York: Viking Penguin.

HEMENWAY, P. 2005. *Divine Proportion: Phi In Art, Nature, and Science.* New York: Sterling.

KAZEMI, D. 2013. *Spelunky Generator Lessons Part 2: Generating the Rooms.* [weblog post]. *Tiny Subversions.* http://tinysubversions.com/spelunkyGen2/. (19 December 2013).

PERRY, L. 2002. Modular Level and Component Design.Or: How I Learned to Stop Worrying and Love Making High-Detail Worlds. *Game Developer* (November 2002). pp. 30-35.

PORTNOW, J. & FLOYD, D. 2012. Extra Credits: Graphics vs. Aesthetics. [online video]. http://www.youtube.com/watch?v=5oK8UTRgvJU. (15 November 2013).

PRIESTLEY, R. 1987. *Warhammer 40,000: Rogue Trader.* 3rd imprint. Nottingham: Games Workshop Ltd.

ROHRER, J. 2009. *Beyond Single Player*. Proceedings from Game Developers Conference, San Francisco, CA. 23 March 2009.

SIGLER, L. E. (trans.) 2002. *Fibonacci’s Liber Abaci. Leonardo Pisano’s Book of Calculation.* New York: Springer-Verlag.

SPUFFORD, F. 2003. *Backroom Boys: The Secret Return of the British Boffin.* ebook ed. (2010). London: Faber & Faber.

WARDRIP-FRUIN, N. 2009. *Expressive Processing: Digital Fictions, Computer Games, Software Studies*. Cambridge, MA: MIT Press.

# Games References

ADAMS, T. 2006. *Slaves to Armok: God of Blood Chapter II: Dwarf Fortress.* [digital download]. Microsoft Windows / Mac OS. Silverdale, WA: Bay 12 Games.

ALLISON, M. 2010. *NetHack 3.4.3.*[digital download]. Microsoft Windows / Mac OS X. http://www.nethack.org.

BLACK, L. 1960. *Black Path Game*. [board game]. Detailed in (Conway et al., 1982, p. 682). Unpublished.

BISKUP, T. 1994. *Ancient Domains of Mystery (ADOM).* [postcardware]. Shareware distribution. http://www.adom.de.

BLIZZARD NORTH. 2000. *Diablo II.* [CD-ROM]. Microsoft Windows / Mac OS. Irvine, CA: Blizzard Entertainment, Inc.

BRABEN, D. & BELL, I. 1984. *Elite.* [cassette, floppy disk]. BBC Micro. Cambridge, UK: Acornsoft.

CELLAR DOOR. 2013. *Rogue Legacy.* [digital download]. Microsoft Windows / Mac OS. Toronto: Cellar Door Games.

CHUNSOFT. 1993. *Torneko no Daibōken: Fushigi no Dungeon. (Torneko’s Great Adventure: Mysterious Dungeon).* [16Mb cartridge] Super Famicom. Shinjuku, Tokyo: Chunsoft.

CUTLER, A. & ASTRAND, A. 1990. *Angband.* [UNIX binary]. Usenet distribution.

FENLASON, J. & WOODLAND, K. 1985. *Hack.* [UNIX binary]. Usenet distribution.

GAMES WORKSHOP. 1989. *Advanced Heroquest.* [board game]. Nottingham: Games Workshop Ltd.

GASLAMP GAMES. 2011. *Dungeons of Dredmor.* [digital download]. Microsoft Windows / Mac OS. Victoria, British Columbia: Gaslamp Games, Inc.

GYGAX, G. & ARNESON, D. 1972. *Dungeons & Dragons Roleplaying System.* [RPG rulebooks]. Lake Geneva, WI: TSR Hobbies, Inc.

HENZEL, L. 1997. *Linley’s Dungeon Crawl (Crawl).* [DOS binary]. GPLv2+.

IDV. 2009. *SpeedTree for Games.* [DVD-ROM]. Game middleware. Columbia, SC: Interactive Data Visualization, Inc.

IWANTANI, T. 1980. *Pac-Man.* [arcade machine]. Tokyo: Namco Limited.

JACKSON, S. 1986. *The Warlock of Firetop Mountain.* [board game]. Nottingham: Games Workshop Ltd.

JOUBERT, R. 2010. *Desktop Dungeons.* [digital download]. Analysis of alpha version. Unpublished.

LAMORISSE, A. 1959. *Risk: A Game of Global Domination.* [board game]. Leeds: Waddington’s House of Games.

LOONEY, A. 1996. *Fluxx.* [card game]. College Park, MD: Looney Labs.

MAXIS. 2000-2013. *The Sims. (Franchise).* [CD/DVD-ROM]. Microsoft Windows. Redwood, CA: Electronic Arts, Inc.

MCMERCHIE, T. 2004. *Tsuro.* [board game]. Rouen, Normandy: Abysse Corp.

MIYAMOTO, S. 1985. *Super Mario Bros.* [8Mb cartridge]. Nintendo Entertainment System. Kyoto: Nintendo Co., Ltd.

NEXTECH. 2002. *Shining Soul.* [GBA cartridge]. Nintendo Game Boy Advance. Ōta, Tokyo: Sega Corporation.

PERSSON, M. 2009. *Minecraft*. [digital download]. Java platform. Stockholm: Mojang AB.

SONOV. 2009. *N.E.O Online.* [digital download]. Unreleased MMORPG. Microsoft Windows. Seoul: Sonov Entertainment.

STEPHENSON, M. 1987. *NetHack.* [UNIX binary]. Usenet distribution.

SQUARE. 1997. *Chocobo no Fushigina Dungeon. (Chocobo’s Mysterious Dungeon).* [CD-ROM]. Sony Playstation. Tokyo: Square Co., Ltd.

SQUARE ENIX. 1987-2013. *Final Fantasy. (Franchise).* [CD/DVD-ROM]. Nintendo Entertainment System, Sony Playstation. Tokyo: Square Co., Ltd.

SUBSET GAMES. 2012. *FTL: Faster Than Light.* [digital download]. Microsoft Windows / Mac OS. Shanghai: Subset Games.

TOY, M. & WICHMAN, G. 1980. *Rogue.* [UNIX binary]. BSD UNIX distribution 4.2.

YOYO GAMES. 1999. *Game Maker (Studio).* [digital download]. Proprietary software: tools. Dundee: YoYo Games Inc.

YU, D. 2009. *Spelunky.* [digital download]. Microsoft Windows. Pasadena, CA: Mossmouth LLC.

# Figure References

Figure 1

JANEK, M. 2013. *Minecraft Waterfall Landscape Wallpaper.* [Online image]. http://mcg.syphrallc1.netdna-cdn.com/wp-content/uploads/2013/05/minecraft-waterfall-landscape.jpg. (21 December 2013).

Figure 2

SERGET3. 2011. *spelunkygameover.* [Online image]. http://www.toptiertactics.com/wp-content/uploads/2011/05/spelunky.jpg. (21 December 2013).

Figure 3

WATTS, J. 2005. *Sample layout 1.* [Online image]. http://cf.geekdo-images.com/images/pic107634\_md.jpg. (21 December 2013).

Figure 4

DERFELLINK. 2012. *Re: New Quest Boards* on *Old Scratch’s HeroQuest Forum.* [Online image]. http://digilander.libero.it/hyw3l/ahqpreview2.png. (29 December 2013).

Figure 5

CARSLAW, G. 2013. *Making up the rules as we go along.* [Online image]. http://3dtotalgames.com/wp-content/uploads/2013/12/fluxx-300x225.jpg. (21 December 2013).

Figure 6

RIZZUTI, B. 2010. *How to Use Create A Sim: Finishing Touches.* [Online image]. http://i3.squidoocdn.com/resize/squidoo\_images/800/draft\_lens8979511module82472541photo\_1265050146create-a-sim-23-thumb.jpg. (29 December 2013).

Figure 7

MCGREGOR, J. 2013. *Cloudberry Kingdom: Platforming Levels Up.* [Online image]. http://oyster.ignimgs.com/wordpress/stg.ign.com/2013/03/521-610x343.jpg. (21 December 2013).

Figure 8

ZANGETSU, T. 2010. *FFIX Morrid’s Tower WM.* [Online image]. http://static2.wikia.nocookie.net/\_\_cb20100402195423/finalfantasy/images/1/10/FFIX\_Morrid's\_Tower\_WM.png. (29 December 2013).

Figure 9

HATFIELD, J. 2007. *Final Fantasy IX (U) (Disc 1: SLUS-01251) Yet more screen shots =).* [Online image]. http://www.ffcompendium.com/misc/ff9battle.jpg. (29 December 2013).

Figure 10

UNKNOWN. 2011. *Desktop Dungeons in HTML5.* [Online image]. http://www.ducknextdoor.com/blog/wp-content/uploads/2011/12/desktop-dungeons-fighter.jpg. (20 December 2013).

Figure 11

MATTHAU, W. 2012. *Legendary thief KazaaakplethKilik Random Event.* [Online image]. http://static3.wikia.nocookie.net/\_\_cb20121102194543/ftl/images/0/00/2012-11-01\_00004.jpg. (18 December 2013).

Figure 12

JACK, M. 2013. *The Brief(?) Journeys of the Junky Eagle* on *RPGnet Forums.* [Online image]. http://i32.photobucket.com/albums/d48/mister\_jack/601\_map1\_zps989dce63.jpg. (18 December 2013).

Figure 13

KEY, E. 2013. *Proteus Screenshots.* [Online image]. http://www.visitproteus.com/presskit/proteus/images/Proteus-2013-01-04-[96%3BuP%3BuB%3B63525082]-0002.png. (20 November 2013).

Figure 14

COFFEY, D. 2011. *The Myth of the Modern Role-Playing Game.* [Online image]. http://www.thegameeffect.com/contentMedia/images/users/DylanCoffey/47b072af-f657-4141-9182-130f78367c70.gif. (21 November 2013).

Figure 15

MARVEL. 2008. *ASCII version – Gameplay.* [Online image]. http://rogue-life.ourden.org/files/images/nethack-shot-2.png. (16 November 2013).

Figure 16

WIKI COMMONS. 2005. *File:LogisticMap BifurcationDiagram.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/e/ea/Logistic\_map\_bifurcation\_diagram.png. (18 December 2013).

Figure 17

WIKI COMMONS. 2013. *File:Mandel zoom 00 mandelbrot set.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/thumb/2/21/Mandel\_zoom\_00\_mandelbrot\_set.jpg/800px-Mandel\_zoom\_00\_mandelbrot\_set.jpg. (18 December 2013).

Figure 18

PATEL, A. 2010. *Polygonal Map Generation for Games.* [Online image]. http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/voronoi-map-goal-16000-shaded.png. (18 December 2013).

Figure 19

WIKI COMMONS. 2005. *File:Fractal weeds.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Fractal\_weeds.jpg/800px-Fractal\_weeds.jpg. (18 December 2013).

Figure 20

WIKI COMMONS. 2005. *File:Dragon trees.*[Online image]. http://upload.wikimedia.org/wikipedia/commons/7/74/Dragon\_trees.jpg. (18 December 2013).

Figure 21

UNKNOWN. 2013. *Modeller Large.* [Online image]. http://www.speedtree.com/images/modeler\_lrg.jpg. (18 December 2013).

Figure 22

WIKI COMMONS: DICKAU, R. 2008. *Cantor 3D3.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/0/0b/Cantor3D3.png. (18 December 2013).

Figure 23

WIKI COMMONS. 2008. *Herr von Koch.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/3/3c/Herr\_von\_Koch.GIF. (18 December 2013).

Figure 24

WIKI COMMONS. 2005. *File:Perlin noise.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/d/da/Perlin\_noise.jpg. (18 December 2013).

Figure 25

WIKI COMMONS. 2007. *File:Perlin.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/8/8b/PerlinNoise2d.png. (18 December 2013).

Figure 26

CALORE, M. 2010. *Conway’s Game of Life in HTML5.* [Online image]. http://www.webmonkey.com/wp-content/uploads/2010/07/LifeHTML5.png. (18 December 2013).

Figure 27

WIKI COMMONS, 2007. *File:Golden ratio line 2.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/thumb/7/77/Golden\_ratio\_line2.svg/270px-Golden\_ratio\_line2.svg.png. (18 December 2013).

Figure 28

WIKI COMMONS. 2007. *File:Golden rectangle and its elements.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/8/84/Golden\_rectangle\_and\_its\_elements.svg.png. (18 December 2013).

Figure 29

WIKI COMMONS: BREW, J. 2005. *File:Spiral aloe.* [Online image]. http://upload.wikimedia.org/wikipedia/commons/3/36/Spiral\_aloe.jpg. (18 December 2013).

Figure 30

SELF-MADE. 2014. *Invader bit-grid.* [Illustration].

Figure 31

ALBERT, I. 2012. *Super Mario Bros. Maps: Level 1-1.* [Edited online image]. http://ian-albert.com/games/super\_mario\_bros\_maps/mario-1-1.gif. (3 January 2014).

Figure 32

EWALT, D. M. 2011. *Pac-Man Fever.* [Edited online image]. http://blogs-images.forbes.com/davidewalt/files/2011/04/pac-man-arcade-screen-822x1024.jpg. (3 January 2014).

Figure 33

KAZEMI, D. 2013. *Spelunky Generator Lessons: Part 2.* [Edited screen capture]. http://tinysubversions.com/spelunkyGen2/. (3 January 2014).

Figure 34

MOLTENBREY, K. 2003. *Computer Graphics World: Familiar Faces.* [Online image]. http://www.cgw.com/images/Media/PublicationsArticle/118371.jpg. (29 December 2013).

Figure 35

EA GAMES, 2013. *Meet The Sims 4!* [Online image]. http://thesimsofficialmag.com/images/issue4/body/sims42.png. (29 December 2013).

Figure 36a

UNKNOWN MODDER. 2007. *Rengal Gown – No Dangly Back.* [Online image]. http://thumbs.modthesims2.com/img/9/2/7/1/6/MTS\_tiggerypum-575051-tig-rengal-nodangle.JPG. (29 December 2013).

Figure 36b

UNKNOWN MODDER. 2007. *MTS2\_Dr\_Pixel\_549480\_NewPJ.* [Online image]. http://thumbs2.modthesims2.com/img/1/3/0/6/7/MTS2\_Dr\_Pixel\_549480\_NewPJ.jpg. (29 December 2013).

Figure 36c

UNKNOWN MODDER. 2007. *MTS2\_Dr\_Pixel\_549680\_newPJ.* [Online image]. http://thumbs2.modthesims2.com/img/1/3/0/6/7/MTS2\_Dr\_Pixel\_549680\_newPJ.jpg. (29 December 2013).

