Combinatorial Explorations
A Brief History of Procedurally-Generated Space in Videogames

Mark J.P. Wolf

The worlds of videogames have grown from single screens of graphics to vast worlds, some of which are too large to ever be fully explored by a single person despite many hours or even years of gameplay. As a quote from Michael Toy – one of the authors of *Rogue* (A.I. Design 1983) – indicates, new experiences are what make games replayable, and keep players interested in a game; and exploration and navigation are among the most basic kinds of experiences afforded by videogames:

> 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. The game was just as hard to win the second time as the first (qtd. in Anonymous 2009).

The procedural generation of videogame spaces not only keeps a game fresh for players, but even for the game's creators, who merely determine the parameters of algorithms which will automate the production of game space. In one sense, the virtual spaces in which videogames' events take place are all procedurally-generated, since they do not exist without the aid of the electronics which produce them; by 'procedurally-generated,' then, we mean the production of significant game content which varies from game to game, and which changes gameplay. At the same time, however, exchanging handcrafted, human-designed locations for algorithmically-generated ones does have certain drawbacks and limitations of its own.

The first procedurally-generated content in videogames could be considered games with randomized content; in *Spacewar!* (Russel 1962), for example, the hyperdrive feature would make the player's ship disappear and then reappear elsewhere at a random location. This demonstrates one of the difficulties in defining procedurally-generated space; for example, if the locations of stars in a star-

field are randomly generated, and the positions of the stars are what define differences in a game's spaces, then one could argue that different spaces are being generated procedurally.

The first game with unambiguously procedurally-generated space, then, would be *Rogue*, one of the most popular mainframe games of the early 1980s. Inspired by William Crowther's text-based *Colossal Cave Adventure* (Crowther/Woods 1977) and the table-top role-playing game *Dungeons & Dragons* (Gygax/Arneson 1974), *Rogue* was a series of dungeon rooms that the player wandered through, defeating monsters, collecting treasure, and looking for food. In order to provide variety, the rooms and the pathways connecting them were procedurally-generated; according to another of the game's authors, Glenn Wichmann (qtd. in Anonymous 2009),

> [w]e originally wanted something very freeform, where a room could be anywhere, and there could be any number of rooms. We couldn't figure out how to do it. We ended up settling on a nine-room tic-tac-toe grid. Then there was the 'mars bug' – sometimes rooms just would not connect. It took us a long time to figure that one out, and we ended up with a number of frustrated players who were having great games and suddenly could not go to the next level because there was no way to get to the staircase.

The randomness present, then, was still within rather tight parameters, and the randomized placement of monsters, treasures, and food also contributed greatly to the replayability of the game. Another game dependent on randomized elements and layout was *Stellar Track* (Atari 1980) for the Atari 2600. This *Star Trek*-inspired resource management game had the player jumping from quadrant to quadrant, using phasors and photon torpedoes to destroy alien vessels. Each of the 36 quadrants was made up of 64 sectors, creating a playing field grid of 48-by-48 positions. Each position could contain the player's ship, or an enemy ship, or a star, which acted as a barrier to travel. Because the placement of stars was randomized in every game, along with the positions of enemy ships and refueling star-bases, each game was a new challenge, again due to randomized content.

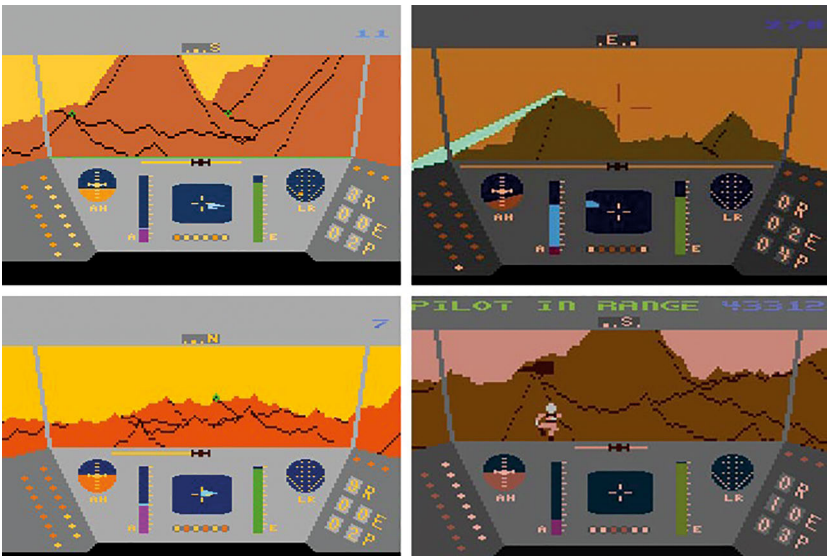
Randomized content was the key not only to making each gaming experience unique and different, but also to make much larger worlds than what could be fit into relatively small amounts of computer memory. The landmark space trading game *Elite* (Braben/Bell 1984) featured a universe of eight galaxies each with 256 planets, in a program of only 22 kilobytes (Noyes 2006). Each planet's position, composition, commodity prices, and name was procedurally-generated, from numeric seeds fed through an algorithm. But while the starships and other objects were visible from space, one did not get a sense of exploring a more earthlike loca-

tion, one with vast tracts of explorable land. This would come with the advent of fractal landscapes.

Fractal Landscapes

In 1975, Benoit Mandelbrot discovered fractal geometry, a branch of geometry that involved shapes that were self-similar at different scales, and that appeared to be able to generate forms like those of natural phenomena (such as fern branches and mountain ranges, which involve copies of the same patterns at different scales). Three-dimensional game graphics were improving and moving from wireframe graphics to filled-polygon graphics, which could be used to present landscapes with a first-person perspective. Emerging from Industrial Light and Magic's work with fractal graphics for the 'Genesis Effect'-sequence in *Star Trek II: The Wrath of Khan* (Meyer 1982), *Rescue on Fractalus!* (Lucasfilm Games 1984) used procedurally-generated fractal landscapes (fig. 1); terrain with geometric mountains of varying heights and randomized terrain helped to create more detailed landscapes than the low-resolution graphics were otherwise able to suggest, as well as their movement in three-dimensional space as the player's vantage point flew across the surface, looking to land and rescue stranded pilots while avoiding aliens and alien fire. Fractal technology would again be used in *Koronis Rift* (Lucasfilm Games 1985) which featured rovers driving over fractal landscapes which were essentially mazes, and *The Eidolon* (Lucasfilm Games 1985a), which inverted its fractal mountains to create a mazelike cave interior. *The Sentinel* (Crammond 1986) also used simple fractal-based landscapes and boasted 10,000 levels, stored in less than 70 kilobytes, and *Starflight* (Binary Systems 1986) used fractal landscapes for its 800 different planets.

Fig. 1: Screenshots from various versions of Rescue on Fractalus!



While initially offering the thrill of three-dimensional landscapes that varied from place to place, the landscapes of these early games were basically just fields of polygons set at different angles to produce a surface with simple mountains, variations which could quickly grow tiresome in their similarity – quantity over quality, as some critics have pointed out (Priestmann 2013; Cham 2014). By contrast, the procedurally-generated two-dimensional underground tunnels seen in a cutaway side view in *Exile* (Irvin/Smith 1988) seem more varied and interesting, and they also include customized hand-designed sections, something which would have been very difficult to include in fractal-based landscapes. There is likewise more direct interaction with the game's spaces in *Exile* than in the other games, demonstrating that despite their graphical detail and aesthetic value, their functionality as interactive elements was severely limited, at least within the limitations imposed by the relatively small amounts of memory and slow processing speeds available at the time.

Also, fractal terrains with their varying slopes can limit accessibility and randomly make certain areas impassable, and it is difficult to ensure that enough of a map will be accessible without limiting mountain heights and valley depths which would homogenize a landscape. Over the years since the early games discussed here, a number of different methods for generating fractal terrains have been developed, and a survey of these techniques can be found in *A Survey of Procedural Terrain Generation Techniques Using Evolutionary Algorithms* (Raffe et al. 2012). The authors compare different techniques, along with their advantages and disad-

vantages, and appropriateness to the needs of different games genres and their needs concerning terrain (for example, flight simulators do not need to ensure accessibility, but may instead require terrain that looks realistic from an aerial perspective).

The uniqueness of fractal landscapes is merely mathematical in nature; handcrafted locations, by contrast, often have specific attributes that together create a distinct personality that turn a space into a place. Player activities in spaces left to procedural generation also tend to be more repetitive in nature; theme and variations, rather than new themes. Prices, commodities, and names may change, but trading, shooting, and being shot at remains pretty much the same. Of course, such games can still be fun; and they tend to be more replayable, with individual games also taking much longer than in hand-designed worlds, due to their greater expansiveness. For some types of games, then, the tradeoff is a worthwhile one. *Captive* (Mindscape 1990), for example, had procedurally-generated planets and bases (65,535 of them), and the game received 91% ratings – by *Amiga Format* (in December 1990) and *Zzzap!64* in (January 1991) – and was listed as the 31st-best game of all time by *Amiga Power*. Also, games like *Elite*'s sequel, *Frontier: Elite II* (Braben 1993), were so detailed, with three-dimensional graphics and spaces, along with the possibility of freeform play, that they were competitive with games using handcrafted locations.

Another reason for procedural generation was due to the memory restrictions of earlier games, when available memory was still measured in kilobytes. With the arrival of optical disc storage around the late 1980s and early 1990s, graphics improved, and handcrafted worlds became much larger – *Myst* (Cyan 1993) is the best example of a successful handcrafted world of the time. While procedural terrain generation would continue to be used, increased storage capacity would allow more flexibility, and more detail could be stored and used, allowing more combination of handcrafted content and procedural generation.

Greater Storage Capacity and Faster Processing Times

With optical discs and hard drives with greater storage capacity, more RAM, and faster processing times, procedural generation would be able to provide more detail in real time, and in an increasing number of areas. MicroProse's *Civilization* series of games, which began in 1990, used procedural generation for the production of maps, allowing players to choose between general types of terrain, like continents, archipelagoes, pangaeas, and so forth. The smaller scale details change, while the overall form remains consistent with the type of terrain chosen, and it is these variations in small details that keep the games fresh. But procedural gener-

ation is not appropriate for every area of a game, since quite often such material lacks the deliberate design that one finds in material handcrafted by an author.

When procedurally-generated content is combined with handcrafted material, the results can be large worlds which have more a deliberate feel in their design. In the mid-1990s *Advanced Dungeons & Dragons: Slayer* (Lion Entertainment 1994) and *Virtual Hydlide* (Technology and Entertainment Software 1995) were released, each of which had worlds that combined handcrafted material with procedural generation. *Slayer* used repeated wall, door, and window elements in a procedurally-generated dungeon which the player explored from a first-person perspective, and which included other features like moving platforms activated by wall buttons and a map that could be consulted, and that filled in gradually as the dungeon was explored (and of course, monsters, torches, and objects were randomly encountered along the way). Players could customize the dungeon as well, choosing the number of levels (10 to 20), Monster Numbers (Few, Handful, Lots, Too Many), Treasure Availability (Poor, Comfortable, Rich, Filthy Rich), Poison Strength (Annoying, Sickening, Deadly, Lethal), Food Availability (Starving, Healthy, Well Fed, Stuffed), Monster Theme (Variety, Mundane, Magical, Undead, Bug), Trap Frequency (None, Few, Lots, Too Many), and Potion Availability (Some, Some More, Lots, Tons).

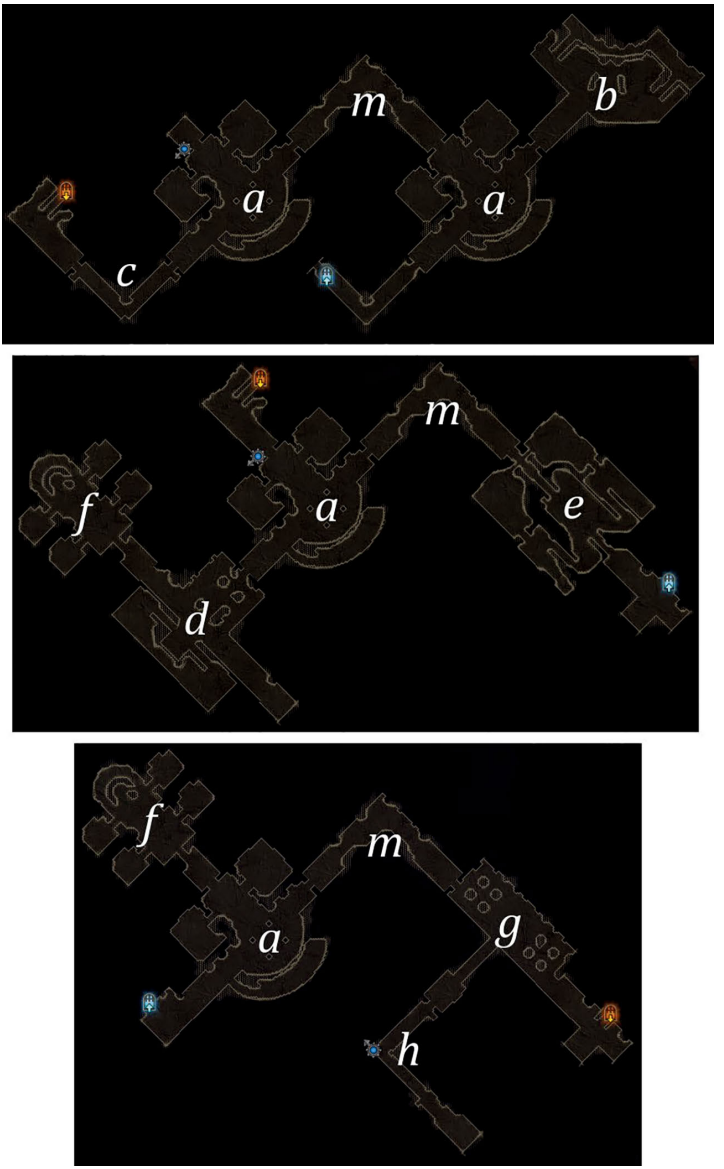
While choosing the settlings for *Virtual Hydlide*, the player is asked to select from "Create world randomly" and "Create world from code," a potentially confusing choice, since code is just as involved with procedural generation as anything else. This choice refers to the fact that the designers of *Virtual Hydlide* created over 20 maps for the dungeon levels and over 20 maps for the overworld, and these maps could be randomly selected and combined to form dungeons, combining handcrafted locations with procedural generation. If a new world was created, the player would be given a ten-letter seed representing that world, which could be reentered later if the player wanted to return to that world. This kind of reusable seed also reminds us that 'procedurally-generated' does not necessarily mean 'randomly-generated,' despite the fact that many such procedures involve randomization. Thus, in addition to the application of randomness to spatial construction, procedural generation can also be seen as a kind of data decompression. A game that I wrote for the TI99/4a computer in the early 1980s, for example, had a cave maze that took 100 screens to display. I managed to compress the data by dividing each screen into twelve tiles (with pathways leaving various sides of the tiles), and each type of tile was represented by a letter so that each of the hundred screens could be represented by twelve letters of text. This would be an example of something procedurally-generated which does not involve randomness, and combines handcrafted locations with procedural generation.

Some games combined various landmasses and buildings to produce procedurally-generated towns. *The Elder Scrolls II: Daggerfall* (Bethesda Softworks 1996)

had a landmass of 62,394 square miles, with 15,000 towns and a population of 750,000. Criticisms and complaints of the monotony of the world (Blancato 2007) led to a much smaller (only 0.01%), but more handcrafted world for its sequel, *The Elder Scrolls III: Morrowind* (Bethesda Softworks 2002). Increased memory capacity can help to solve the problem of repetition in two ways; games can include more handcrafted interchangeable elements to be used in procedural constructions, or a greater number of algorithms can be used to add parameters to randomization as well as generate higher levels of detail. For example, *Pixel City* generates dozens of random buildings in various styles to keep the cities that are generated from looking repetitive (Young 2009). Other city generators, like *Subversion City Generator* (Introvision 2011) also plan city layouts based on waterways, bridges, and major and minor avenues, before filling them with buildings, creating more realistic urban landscapes.

The spaces of individual buildings and dungeons can also be done through the recombination of handcrafted building units. Games from the *Diablo*-series, for example, are noted for this technique. Three different floor plans of "Cathedral"-Level two from *Diablo III* (Blizzard Entertainment 2012) show how handcrafted building sections are recombined to create different arrangements which randomize gameplay (fig. 2a-c). While the repetitions in such designs are more evident from a top view, they may be less so from a first-person point of view, especially if the building interiors have randomized elements and décor.

Fig. 2a-c: Three different procedurally-generated floor plans from Diablo III



Interestingly, another consequence of greater storage capacity is how the constraints that programmers had to work under out of necessity, due to the lack of available memory, have now become something of an artistic challenge. Programmers taking up this challenge attempt to make as detailed worlds as possible fit within small amounts of memory, relying heavily on procedural generation and

the faster processing speeds, which are able to decompress data, do fractal calculations, and run algorithms much faster than did the computers of the 1980s and 1990s. *Noctis* (Ghignola 2000), for example, features a three-dimensional, texture-mapped explorable universe of billions of stars and planets, all generated by a program smaller than one megabyte. Regarding the programming behind the fourth version of *Noctis* from 2003, Ghignola stated in an interview:

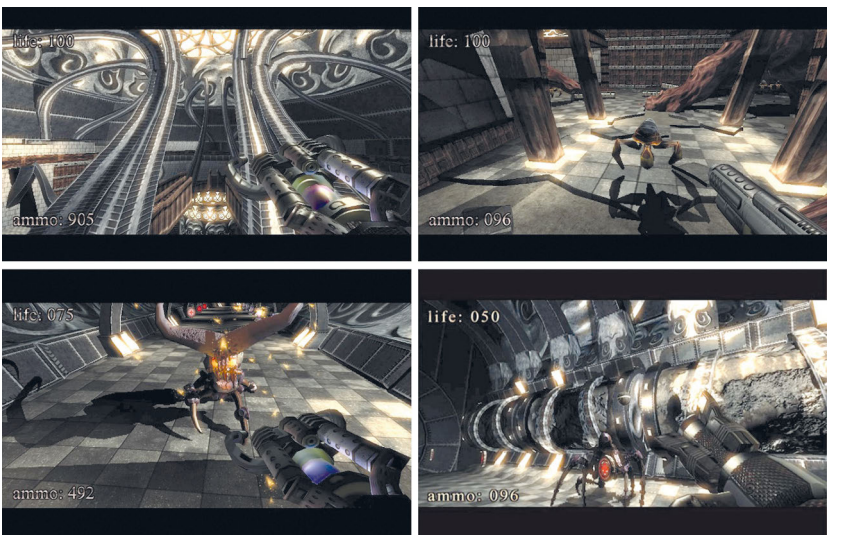
> Its 'engine' is a mixture of sparse libraries. Planets and stars, for example are textured as pre-projected spherical maps. I don't know how many remember Quicktime [sic] VR, but I guess Wikipedia might have an article on that for those who never heard of it. QTVR worked by projecting a scene (a composite photograph) over a virtual, spherical screen, splashed on the physical flat screen. Well, *Noctis* planets work the other way around: they get a rectangular raster image and wrap it around a sphere. This is convenient in terms of speed because, if you can tolerate losing realistic, perspective aberrations when the sphere is significantly off the center of the viewport, the spherical map can be entirely precalculated, resulting in a rendering that was very fast even on my 486 of those times. The reverse (projection of the inside of a sphere) was used for skies on the surfaces of planets. Then there was a polygon engine taking care of drawing the heightmap constituting the surface itself. The polygon engine was pretty simple, but again pretty fast: it didn't even perform depth buffering (I doubt I could find enough memory for the buffer anyway), it just relied on the painter's algorithm and minimal hidden surface removal of one-sided surfaces. It was optimized enough that it could afford texture mapping of arbitrary-angled polygons, at about 1 division every 16 pixels. Where more detail was a good idea, such as to simulate grass on terrains, an additional texture layer was overlaid to the 'ground' texture, in a sort of very simple kind of bump mapping. The 256-color palette was split into four gradients having 64 brightness levels each, which finally enabled blur effects; in particular, the 'vimana drive' effect seen while traveling through interstellar medium was obtained by using an off-center blur filter over a persistent canvas. In practice, the effect was repeated each frame without clearing the previous frame, leaving trails whenever an element moved through the screen. I don't sincerely remember much about the shading of polygon surfaces, but I guess it was plain-color shading, driven by the angle of incidence of light sources. What more? One nice addition was the use of concentric, semitransparent lines to create halos around the light of stars, in such a way that – in my idea of that time – would mimic more the effect of light passing through an organic eye, rather than a camera's lens flare (Szymanski 2012).

Ghignola has revised the program in subsequent releases (and continues to work on the program; *Noctis V* is currently in the works as of mid-2019), and although

the *Noctis* games are not commercially distributed, they have attained a following and are considered art games.

Another example, from Germany, is *.kkrieger* (.theprodukkt 2004), a first-person shooting game with detailed graphics and elaborate interiors (fig. 3a-d). Among its various procedural techniques is box modeling, in which 3-D primitives (like cubes, spheres, or cylinders) are subdivided with each section being replaced by more detail, a repetitive process which bears some similarity to the iterations involved in the production of fractal imagery – according to their website, some of the game's models and textures take hundreds of steps to create when the game is run. Through box modeling, texture mapping, and interactive lighting, the game uses only 97280 bytes of disk space and is able to produce a world which would normally take hundreds of megabytes to store. The small size of the game, however, does not mean that it could have produced at an earlier time when less memory was available. The many steps and processes used by procedural generation algorithms require more processing time, and faster processors for the game to operate; *.kkrieger*'s system requirements include a 1.5GHz Pentium 3/Athlon or faster, 512MB of RAM, and a GeForce4Ti (or higher) or ATI Radeon8500 (or higher) graphics card supporting pixel shaders 1.3, preferably with 128MB or more of VRAM. Thus, the small amount of memory used must be made up for through processing speed.

Fig. 3a-d: .kkrieger



Greater storage is also necessary for games which generate and output large amounts of world data during their world creation. *Slaves to Armok: God of Blood*,

Chapter II: Dwarf Fortress (Tarn Adams 2006) – usually referred to simply as *Dwarf Fortress* – is a work still in progress, which generates all the terrain of its worlds, along with maps for elevation, temperature, rainfall, drainage, vegetation, and salinity. Mountains are eroded by rivers, and plant and animal populations are added, as well as races of sentient beings. Weather modeling even tracks wind and humidity and creates fronts, clouds, storms, and blizzards. Once the material world is generated, a historical timeline is generated, with thousands of characters being born, living lives, and dying, and events being recorded, so that in "Legends" mode, one can find a year-by-year list of the major events for every character's life, where they lived or wandered, who they fought, outcomes of conflicts, descendants, and so forth. Players can choose how much history is generated before the world is ready for their characters to inhabit. Like *Rogue*, graphics are two-dimensional and text-only (the Code Page 437 character set originally used for IBM PC computers) and appear in 16 available colors.

Generating three-dimensional terrain, the racing game *Fuel* (Asobo Studio 2009) provides players with 5560 square miles of fully-explorable terrain, for which the company received a *Guinness Book of World Records*-certificate for the largest playable environment in a console game (Fahey 2009). On one hand, the game is a good example of some of the graphics that procedural generation can achieve, underscoring the fact that it involves much more than simply randomness, but instead a set of parameters within which plausible objects and landscapes are generated; and these rules can be quite complex and elaborate ones, which, when they are balanced and adjusted just right, can create scenes which would be difficult to tell from handcrafted ones – movies have used similar techniques; Pandora, the planet in *Avatar* (Cameron 2009) was largely procedurally-generated. On the other hand, good graphics are not enough to make up for too much repetition of elements. As Marsh Davies noted in EDGE Online:

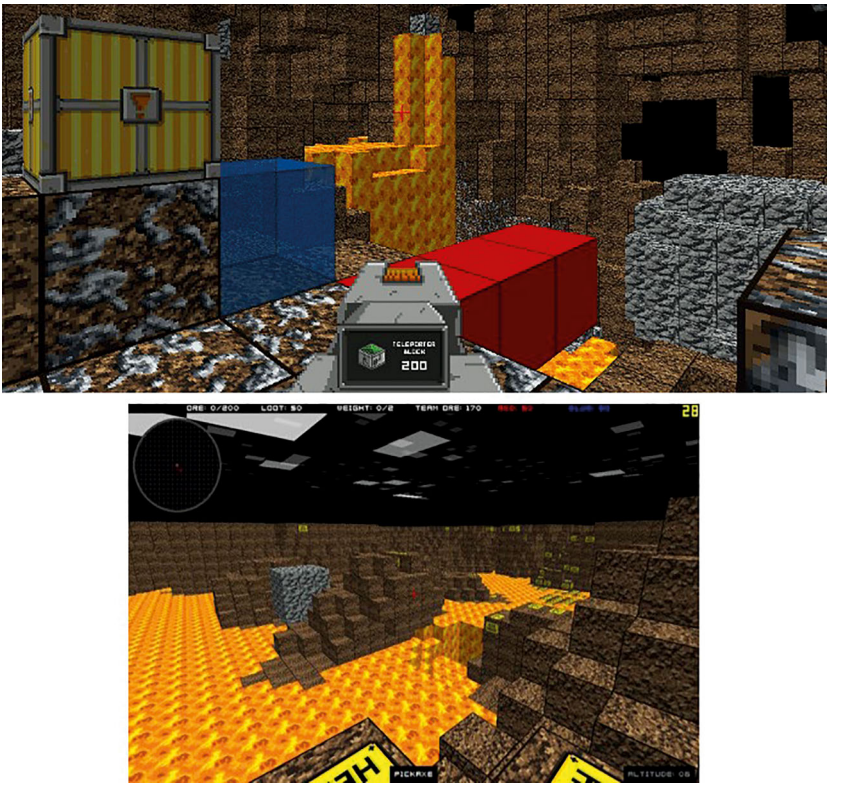
> Importantly, however, realism is not the aim – and cannot be for a world which hopes to serve as the basis for a game. Making a world realistic is not the same as making it interesting, a rule to which a game like Fuel stands in testament. Its recreation of the American wilderness was both beautiful and credible moment-to-moment, but empty and repetitive in aggregation, the terrain never quite feeling suited to the purpose of racing (Davies 2011).

While the procedural generation found in space trading games acted as a kind of backdrop for actions which were themselves typically quite repetitive (buying, selling, trading), three-dimensional worlds usually must be far more interactive for the player. Racing games like *Fuel* allow players to travel all over the landscape exploring, but beyond that, interaction is usually rather minimal. But more fully interactive procedurally-generated worlds were appearing as well.

'Infiniminer,' 'Minecraft' and beyond

The procedurally-generated world to receive the most publicity today is probably Markus Persson's *Minecraft* (Mojang 2009). Inspired by *Dwarf Fortress*, as well as Peter Molyneaux's *Dungeon Keeper* (Bullfrog Productions 1997), and especially *Infiniminer* (Barth 2009), from which *Minecraft* borrows heavily for its ideas, visual design, gameplay mechanics, and procedural-generation techniques. *Infiniminer* (fig. 4) appeared in late spring of 2009, but work on it was discontinued after source code was leaked, and other games besides *Minecraft*, including *Fortress-Craft* (ProjectorGames 2011), *CraftWorld* (2.0 Studios 2011), and *Ace of Spades* (Jagex 2012) also used and modified code from the game. Other games inspired by *Minecraft*, such as *3079* (Phr00t's Software 2011) and *Cube World* (Picroma 2013), have a similar look and feel as well.

Fig. 4: Zachary Barth's Infiniminer



Games like these deal with the problem of repetition by moving it to a smaller scale; instead of the same trees or buildings, their worlds are composed of variations of blocks combined in endless ways. This granularity also allows users to build and destroy more easily, since construction and destruction is simplified down to the appearance or disappearance of blocks. While the graphics of these games are simpler and more stylized (something which the retro movement in gaming has made more acceptable to contemporary audiences who are used to photorealistic graphics), the possibilities inherent in the interactivity available far outweighs the aesthetic tradeoffs for many gamers. As such, space trading and exploration games have continued to evolve, with procedurally-generated planets and other locations in games like *FTL: Faster Than Light* (Subset Games 2012) and *Starbound* (Chucklefish 2016). Perhaps the most ambitious procedurally-generated locations are the millions of planets in *No Man's Sky* (Hello Games 2016). As Sean Murray (qtd. in Parkin 2014), one of the creators, put it:

> We are attempting to do things that haven't been done before… No game has made it possible to fly down to a planet, and for it to be planet-sized, and feature life, ecology, lakes, caves, waterfalls, and canyons, then seamlessly fly up through the stratosphere and take to space again. It's a tremendous challenge.

When older games revealed planets' surfaces, it was little more than a series of fractal mountains and bodies of water; even Noctis only featured rudimentary plants and rock structures. The planets of *No Man's Sky*, however, have lighting conditions based on the type of nearby star and positioning in orbit, and plants and animals involved in ecosystems, all of which are animating as you fly through the scenery, a level of detail previously only seen in handcrafted environments in games. Because the universe generated for the game is unexplored, even making a demo for the game posed problems not usually encountered in other games. According to Murray (2014):

> To give you an example of some problems, we planned out what our demo was, and then we had to find somewhere in the universe to set it. So I flew around for quite some time, a couple of days, looking for a planet that particularly suited it. So I had to pick that planet, but also find another planet that was nearby that I was going to fly to, and kind of engineer this situation where there was going to be things to fight in between. And then you actually end up having to deal with really weird things like the time of day on the planet it starts from, and what animals are going to be out at the time of day, and what time of day it is on the planet you land on. We wanted that to be daytime, and that's really hard to plan, and it just doesn't happen that easily. It was quite a fun little problem to have. Whereas, for any other game, you would be constructing something for months especially for E3. When

> Ubisoft shows off what *Assassin's Creed* is like, it has specifically made that entire demo for that show. We don't have that control, which is really good, but also really crazy.

Other works-in-progress promise detailed universes of near-infinite size, such as Shamus Young's *Project Frontier*, Miguel Cepero's *Procedural World*, and Josh Parnell's *Limit Theory*, each with their own aesthetics and approach to procedural generation. What all these games suggest is that when it comes to the creation of vast videogame spaces, and the content within that space that defines it, we are seeing a shift in which 'handcrafted' will not refer to specific content or particular instances of objects and locations so much as to ranges of possibilities and sets of parameters within which many variations of the required content can be generated. What parameters cover, how they are set, what settings are possible, and how different sets of parameters are linked to each other, determining ranges and limiting outcomes, will be the main ways of combining human hand-crafting with algorithmic construction and randomness. Open-world sandbox games have already made other games seem more limited in their interactivity, but so far the narratives they generate during gameplay are much weaker due to the wide range of possibilities to be accounted for. But as artificially-intelligent agents improve, and potential of story structure and world structure are explored and realized, we will likely see the quality of emergent narratives rising as well.

For games to move in the direction of procedurally-generated content is quite natural when one considers how much of the complexity of the physical universe is due to procedural processes. As fractal mathematics and the study of cellular automata has demonstrated (Wolfram 2002), simple rules and concepts can generate complex structures, like a seed growing into a tree or strands of DNA guiding the development of the human body. Videogame worlds grown by algorithms are increasing in their complexity, and just as players explore these worlds, their designers are exploring the nature of worlds and their representations. While they will never reach the elegance and ingenuity of the procedural processes found in the natural world, their striving to imitate them can make us all the more appreciative of the universe around us and its combined simplicity and complexity.

References

2.0 Studios (2011): *CraftWorld*, Windows Phone: Xbox Live.

A.I. Design (1983): *Rogue*, PC: Epyx.

Anonymous (2009): The Making of Rogue, in: *EDGE*, July, 3rd, edge-online.com/features/making-rogue/2.

Asobo Studio (2009): *Fuel*, PS3/Xbox 360: Codemaster.

Atari (1980): *Stellar Track*, Atari 2600: Sears.
Barth, Zachary (2009): *Infiniminer*, PC: Zachtronics Industries.
Bethesda Softworks (1996): *The Elder Scrolls II: Daggerfall*, PC: Bethesda Softworks.
–––– (2002): *The Elder Scrolls III: Morrowind*, PC: Ubisoft.
Binary Systems (1986): *Starflight*, PC: Electronic Arts.
Blancato, Joe (2007): Bethesda: The Right Direction, in: *The Escapist*, February 6th, escapistmagazine.com/articles/view/video-games/issues/issue_83/471-Bethesda-The-Right-Direction.
Blizzard Entertainment (2012): *Diablo III*, PC: Blizzard Entertainment.
Braben, David (1993): *Frontier: Elite II*, Amiga/Atari ST: GameTek.
Braben, David/Bell, Ian (1984): *Elite*, PC: Acornsoft.
Bullfrog Productions (1997): *Dungeon Keeper*, PC: Electronic Arts.
Cameron, James (2009): *Avatar*, Film: USA.
Cham, Brian (2014): The Next Space: Procedural Generation as Indicator of Technological Possibility, videogamesftvms2014.wordpress.com/tag/procedural-generation.
Chucklefish (2016): *Starbound*, PC: Chucklefish.
Crammond, Geoff (1986): *The Sentinel*, PC: Firebird.
Crowther, William/Woods, Don (1976): *Colossal Cave Adventure*, PDP-10: Crowther/Woods.
Cyan (1993): *Myst*, Macintosh PC: Brøderbund.
Davies, Marsh (2011): Building Worlds with a Single Click, in: *EDGE Online*, July 6th, edge-online.com/features/building-worlds-single-click/3.
Fahey, Mike (2009): Fuel is the Biggest Console Game Ever, in: *Kotaku*, May 22nd, kotaku.com/5265942/fuel-is-the-biggest-console-game-ever.
Ghignola, Alessandro (2000): *Noctis*, PC: Ghignola.
Gygax, Gary/Arneson, Dave (1974): *Dungeons & Dragons*, Pen&Paper: Tactical Studio Rules.
Hello Games (2016): *No Man's Sky*, PS4: Hello Games.
Introvision (2011): *Subversion City Generator*, PC: Introvision.
Irvin, Peter/Smith, Jeremy (1988): *Exile*, BBC/Electron: Superior Software.
Jagex (2012): *Ace of Spades*, PC: Jagex.
Lion Entertainment (1994): *Advanced Dungeons & Dragons: Slayer*, 3DO: Strategic Designs.
–––– (1995): *Virtual Hydlide*, Sega Saturn: Sega.
Lucasfilm Games (1984): *Rescue on Fractalus!*, Atari 5200: Atari.
–––– (1985): *Koronis Rift*, Atari800/C64: Epyx.
–––– (1985a): *The Eidolon*, Atari800/C64: Epyx.
Meyer, Nicholas (1982): *Star Trek II: The Wrath of Khan*, Film: USA.
Mindscape (1990): *Captive*, Amiga/Atari ST: Mindscape.
Mojang (2009): *Minecraft*, PC: Mojang.

Murray, Sean (2014): *No Man's Sky: The Story, Gameplay, and Multiplayer Explained*, youtube.com/watch?v=tYoGN2zgXQU&t=65.

Noyes, Emma (2006): David Braben: From Elite to Today, in: *Gamespot*, November, 22nd, gamespot.com/articles/qanda-david-braben-from-elite-to-today/1100-6162140.

Parkin, Simon (2014): No Man's Sky. A Vast Game Crafted by Algorithms, in: *MIT Technology Review*, July 22nd, technologyreview.com/news/529136/no-mans-sky-a-vast-game-crafted-by-algorithms.

Phroot's Software (2011): *3079*, PC: Phroot's Software.

Picroma (2013): *Cube World*, PC: Picorama.

Priestmann, Chris (2013): Why It's Best to Be Cautious around Procedurally Generated Indie Games, *Indiesstatik*, December 8th, indiestatik.com/2013/12/08/procedural-generation.

ProjectorGames (2011): *FortressCraft*, Xbox 360: Xbox Live.

Raffe, William L./Zambetta, Fabio/Li, Xiaodong (2012): A Survey of Procedural Terrain Generation Techniques Using Evolutionary Algorithms, in: *WCCI 2012 IEEE World Congress on Computational Intelligence*, June 10th-15th, goanna.cs.rmit.edu.au/~xiaodong/publications/ptg-raffe-cec2012.pdf.

Russel, Steve (1962): *Spacewar!*, PDP-1: MIT.

Subset Games (2012): *FTL: Faster Than Light*, PC: Subset Games.

Szymanski, David (2012): Interview with Alessandro Ghignola (aka 'Alex'), in: *Videogame Potpourri*, May 9th, homeoftheunderdogs.net/game.php?id=2950.

Tarn Adams (2006): *Slaves to Armok: God of Blood, Chapter II: Dwarf Fortress*, PC: Bay 12 Games.

.theprodukkt (2004): *.kkrieger*, PC: .theprodukkt.

Wolfram, Stephen (2002): *A New Kind of Science*, Champaign, IL: Wolfram Media 2002

Young, Shamus (2009): *Procedural City*, shamusyoung.com/twentysidedtale/?p=2940.