***Abstract –* The purpose of this paper is to provide an overview and critical review of the various techniques for the procedural generation of game content both proposed in research and examples in commercial games. This topic has seen a large amount of academic interest over the last number of years thus a fully exhaustive review is beyond the scope of this review. Instead it will explore the strengths and weaknesses of the four main categories of content generation and analyse choice examples for each before concluding with a discussion of how this has informed the design of the project.**

1. Introduction

The development of video games has become increasingly more expensive and time consuming as technology has improved, with developers spending millions of dollars and taking years to produce more and more detailed games.[1] This cost is largely caused by both improvements in, and the AAA[[1]](#footnote-2) market's focus on, graphical technology requiring more and more assets to be created by the artists.

Similarly designing games has proved more challenging due to the wide range of play styles and expectations of gamers; even within a single game, as shown by Drachen *et al's.* study of Tomb Raider Underworld[2] that found four distinct play styles within the relatively restricted game. Accommodating all of these players and crafting an enjoyable experience for as many play styles as possible is a daunting task.

One approach which can be applied to both of these problems and has seen large scale adoption within the game's industry is the procedural generation of content. The most immediate benefit to generation is the ability for smaller teams to produce expansive, content rich games within the same timeframe as a much larger team. It can also be used as a creative aid to augment a designer's imagination and is even a core game mechanic in increasing numbers of games; including Minecraft, one of the most successful games of all time. [5]

2. Content Generation Techniques

Procedural Content Generation(PCG) is the collective term for a large range of techniques to automatically create content with minimum human input. In this context, *'content'* can be defined as any non agent (Non playable Character) behaviour systems within the game i.e. textures, terrain, items, or even entire plot lines [3].

Togelius *et al.* in their taxonomy of the subject offer a further distinction of whether the content is *'necessary or optional'*[4]. Necessary content is aspects of the game the player must interact with in order to progress while optional is content which can be entirely ignored by players with no great impact to their experience.

Togelius *et al.* also define several features to aid in the comparison of PCG approaches;

1. *'Online or Offline'*: Whether the content is generated at the runtime of the game or during development of the game.
2. *'Random Seed or Parameter Vectors'*: A detail of the algorithm itself, this property describes the number of parameters, and thus amount of control, the algorithm allows the developer or whether these parameters are 'seeded' using random numbers.
3. *'Stochastic or Deterministic':* The amount of randomness within the generation, i.e. a purely deterministic algorithm with generate identical content if repeatedly called with the same inputs.

Due to these sometimes conflicting properties a large array of algorithms for PCG have been developed. Despite this breadth most can be grouped into broad categories; Constructive, Generate and Test and Search Based. These will be discussed in depth in the following sections along with a separate section for Narrative generation which comes with it's own unique challenges and approaches.

2.1 Constructive

Constructive PCG refers to generation processes that execute once in order to produce content. This means the algorithms must be guaranteed to produce usable content of an acceptable quality every time.

The earliest examples of constructive PCG were purely deterministic and used as a form of memory optimisation. As a work around of the hardware constraints of early computers, content was saved in a compressed format which could then be expanded out at runtime. A prime example of this was *Elite* [6] which contained 8 galaxies of 256 planets that were completely generated (name, position, items etc.) from a seed number. A more modern example is *.kkrieger*[7] which compresses an entire first person shooter into a 97kb file. Advances is hardware and the large impact on load times has made the use of PCG for data compression almost entirely abandoned by the industry.

The most prevalent use of constructive algorithms in games is the generation of terrain. A variety of methods exist to produce fractal like landscapes by stochastically manipulating vertex elevation to produce a hightmap representation of the terrain. One of the earliest and more popular implementations is Fournier *et al.* Diamond-Square Algorithm.[9] Introduced in 1982, the algorithm takes a square 2D array with a width of 2^n + 1 and recursively subdivides the array into diamonds and squares setting the midpoint to the average of each corner plus a random number until all elements are set. At each iteration the magnitude of the random number is reduced to produce gradual slopes. While the Diamond-Square algorithm can produce good looking and realistic terrain in a reasonably efficient way, a number of flaws have been pointed out. Miller [10] criticised the algorithm due to the noticeable creases produced as the more significant changes in height occur in a rectangular grid. Diamond-Square results can improved to remove these artifacts by using a generalised stochastic algorithm to calculating the midpoint using a linear estimation rather than averaging the neighbours[11]. Miller also proposed the Square-Square algorithm, outlined in [10], which aimed to avoid the crease like artifacts.

A key flaw with most sub-division techniques is that they offer little control over the generation process beyond providing a roughness constant which scales the random displacement of the midpoint at each iteration to control the steepness of the displacement and the initial seed values. Another disadvantage to sub-division is that it is limited to one or two dimensions meaning that overhangs and cave structures cannot be generated.

An alternate approach to creating fractal terrain to generate noise for elevation data. By far the most popular noise generation is Perlin Noise [12] and its variants [13]-[15]. It was first introduced in 1985 as an attempt to produce more natural looking textures using gradient noise. The original Perlin algorithm defines an n-dimensional grid with each node assigned a random unit vector. For any given point the dot product of the surround node's gradient vector and the distance to that point are calculated and a fade function applied to interpolate across these values. This leads to very poor scaling due to the complexity O(2n) though subsequent versions resolve this. [13]

The ability to use n-dimensions allows for the generation of overhang and cave structures, unlike sub-division, by using three or more dimensional noise. Minecraft[16] demonstrates this use of Perlin[17]. This allows for more diverse and interesting terrain than can be achieved through sub-division.

Depending on the generation requirements the choice between sub-division or Perlin Noise is largely a person choice. For one and two dimensions Diamond-Square is simpler to implement and faster[18] it can lack the variance in terrain that Perlin produces and more easily suffers from linear artifacts. However if cave like structures are required Perlin or its varients are the only choice. Both algorithms can produce interesting terrain but struggle to produce near photo realistic terrain on their own. Similarly, Howard Zhou *et al.*[19]highlights the lack of control both provide. Users are unable to specify the type or placement of terrain features; e.g. the ability to specify a river must exist in the terrain.

It has been noted [21][22] that due to fractal terrains isotropic nature it cannot accurately describe several aspects of nature. Musgrave *et al.* [22] were the first to suggest modelling physical erosion to produce more natural terrain. They proposed two separate algorithms to model hydraulic and thermal erosion. The hydraulic algorithm works by storing the current height, water and sediment values for each point, over a number of time steps a fraction of the water and sediment is distributed to the neighbours with a lower elevation. If the water value reaches 0 the sediment value is added to the height value of that point. The thermal algorithm is a simpler model that smooths large changes in height. At each timestep the difference in elevation between a point and its neighbour is compared to a global talus angel. If the difference between points is greater than the talus angel a fixed percentage of the point is transferred to the neighbour. Various improvements have been suggested [23][24][25] for the original hydraulic algorithm seeking to produce more physically realistic simulations. However, Musgrave's original was already computationally expensive thus making their approaches less suitable for online generation. Benes explains in [26] that hydraulic algorithm can produce noticeable oscillations and provides little controllability. He broke Musgrave's algorithm into 4 separate simplified steps, water placement, evaporation, deposition and water transport. While not physically accurate Benes approach creates a reasonably approximation in a faster time. In [27] Olsen provides an optimisation and analysis of both Benes and Musgrave's work in the context of their use in video games. While the optimised versions of both the hydraulic algorithm drastically reduced in runtime while maintaining similar erosion scores (standard deviation of the maps slopes divided by the mean value[22]) it were still too slow for runtime generation with 500 iterations taking an average 25s (Not including base terrain generation). Olsen solution was to change Musgrave's thermal algorithm to ignore slopes steeper than the talus angle and flatten the rest. The proposed algorithm created higher scoring, using on Musgrave's heightmap quantifying function described in [22], terrain in less iterations, achieving 2.15 in 50 iterations compared to the 450 hydraulic required. It also resulted in linear time scaling with 500 iterations taken an average of 10s. The rendered terrain while less realistic looking is likely more usable within games due to the larger flat areas and smoother height transitions.

Overall, erosion techniques can produce more realistic, smoother terrain than a pure fractal approach though at a large computational expense and often provide little control making them less suitable for run-time use.

Attempting to solve the issue of controllability while maintaining the desirable stochastic nature of PCG Doran *et al.* investigated using software agents to model the terrain in [20]. Their system uses a number of asynchronous, autonomous agents that change the evaluation of points within the heightmap according to their predefined goal. To demonstrates the concept Doran *et al.* created 5 agent types; (1.) coastline, creates the initial landmass. (2.) smoothing, take random walks lowering elevation. (3.) beach, flatten areas near coastline. (4.) mountain, raise mountain chains. (5.) river, erode a random path from high to low elevation. Each agent is relatively simplistic but the interaction between them produce unique results. The system is easily extendible by introducing agents with new goals. The large degree of oversight for designers on the generated terrain is achieved by specifying the number of agents of specific types i.e. selecting 3 river agents results in 3 rivers being generated. However this increased control comes at the price of slower generation speeds. A 512x512 heightmap reportedly took 20 seconds on a 3.2GHz pentium, though it should be noted the paper was published nearly 10 years ago at the time of writing and thus performance has likely improved on modern hardware.

While this section has focused on constructive terrain approaches as this is a major focus of both the industry and academia, terrain is not the only content that can be generated. Tools such as SpeedTree [8] procedurally build the models and texture atlases for foliage. Similarly, NaturalMotion's Euphoria animation engine[63] provides real time generation of character animations by simulating the physical bone and muscular structure.

2.2 Generate and Test

Generate and Test algorithms perform some for of check on the content generated and may perform a number of iterations until content of an acceptable quality has been generated.[4] Unlike constructive algorithms, generate and test is not limited to content that is guaranteed to work potentially leading to more varied results. This comes at the expense of run time as the algorithm could continually fail to create content of an acceptable degree. An example of this approach in action is the world generation in Dwarf Fortress[27] which continually checks and rejects candidate worlds during generation.[28]

2.3 Search Based

Search-Based is a variation on Generate and Test that evaluates the candidate and assigns a gradient fitness value instead of a simple pass or fail. Subsequent candidate generation is contingent on the fitness of previous instances with the aim of producing higher scoring content.[4] In other words rather than continually generating from scratch, the algorithm searches for the best solution. This naturally lends itself, but is not limited to evolutionary algorithms (EA); though a significant percentage of the examples discussed do make use of evolutionary techniques.

EA and the variation Genetic Algorithms (GA) work by maintaining a population of potential solutions where the more fit candidates are combined and or “mutated” to form new generations until a solution of sufficient fitness is found or a maximum number of iterations completed. [29] provides an introduction to the topic. The examples discussed in this section can be further distinguished based on how they evaluate the fitness of the generated content.

Designing a function to correctly evaluate something as subjective as game content is non-trivial, especially when developers and players may value different aspects. Though this can be partly achieved by objectively assessing features of the content. However the most direct solution is to allow users to provide feedback to drive the search. This is known as interactive evaluation and allows players to tailor their gaming experience to their own preferences. Both [30] and [31] using player feedback to evaluate generated maps.

In [30] Liapis *et al.* evolve real time strategy maps represented as 74 value arrays containing the coordinates of player bases, resource nodes and impassible areas. Each map is evaluated on 3 spacial navigation, 3 resource distribution and 4 aesthetic algorithms with the overall fitness given as a weighted sum. The weighting of this sum was adjusted interactively using two methods; players select their favourite map from a pool of candidates, or players rank the pool based on their preference. Experiments using AI to model player preferences showed the algorithm did generate maps tailored to the players preference, with the rank-based method adapting much faster. The rank-based evaluation was also much faster to adapt to changes in player preferences. However both performed poorly compared to the control, non interactive algorithm which optimised in less iterations, thus creating objectively “better” maps, by always selected the map with the highest fitness. Raffe *et al.* [31] investigated a somewhat similar approach but introduced a recommender-system(RS) based player model to evaluate the content. When a new map is to be generated 7 potential map geometries based on the last map are created using pre-build tiles. The player chosen geometry is then passed to a Compositional pattern-producing network neuroevolution of augmenting topologies (CPPN-NEAT)[32] to evolve CPPNs which calculate the position and density of game content such as weapons, enemies etc. The RS player model evaluates the candidate CPPNs and assigns the probably of being enjoyed. After 10,000 iterations or a perfect candidate being found the geometry and CPPN are combined. Once the player has beaten the generated map they rate it between 1 and 5. This rating then updates the RS model for the next iteration. While play testing showed players generally preferred the evolved content over randomly generated maps it was only by a slight margin. Raffe *et al.* were unclear whether this was caused by a poor player model or players self-validating their map choice. Similarly, 75% of participants gave the maps an average rating which Raffe *et al.* suggest might be caused by the CPPN-NEAT reaching a local optimum and repeating content or by the games limited assets.

Interactive evaluation does not necessarily require direct input from the user. [33] [35] and [36] have all experimented with using indirect data collection for player experience modelling to produce personalised content. Player modelling, also known as Experience-Driven PCG, is the process of building a model of the user's preferences and or emotional state to better assess the quality of generated content. Yannakakis and Togelius provide a taxonomy of Experience-Driven PCG in [37] which outlines 3 ways of building the Player Experience Model (PEM).

1. Subjective PEM: Directly ask players about their experiences, as discussed above.
2. Objective PEM: Measure physiological responses to predict emotional state.
3. Gameplay Based PEM: Monitor player input and actions within the game.

Objective PEM is potential very invasive and largely impractical to employ outside of a lab environment. While it has seen use in commercial games, [38][39], it largely appears as a gimmick.

Hastings *et al.* in [33] built *Galactic Arms Race,* whichuses Gameplay Based PEM by measuring which weapons players used to create a preference model to evaluate new candidate weapons. Similarly [35] assigned players to pre-defined models based on the actions within the first level in order to bias the level generator towards their preferences. This approach is the least invasive but interpreting player actions involves making assumptions on the reason behind them. Using *Galactic Arms Race* as an example, a player may use a weapon because it is more powerful, rather than because they enjoy using it as the system would assume.

While allowing players to personalise game content to their taste is an appealing idea and appears to improve the experience for players [33][34] there are problems with the approach. Explicitly gathering player information for interactive evaluation interrupts gameplay and can break immersion while indirect is potentially invasive and can easily produce inaccurate or noisy data. The approaches discussed above generally only provide the player a single screenshot on which to evaluate the candidates or use a simple number rating. The former doesn't provide enough information for the player to easily judge the content which can prevent optimisation by selecting conflicting preferences; while the latter can lead to inaccurate player models.

Another approach to determining the fitness of generated content is to simulate its use in game using a software agent. In [40] Fisher describes his use of an agent continually playing through a generating platformer level to provide feedback to another agent searching a tree representing the level. Shaker *et al.* in [41] outline the complexity of building these agents and show one approach for a deliberative agent used to find the playability of generated Cut The Rope levels. They continued their work in [42] in which they propose evolving generic timelines of interactions to represent game levels. A simulation is then ran on the timeline and constructs the various required level components at each event. In [44] Khalifa *et al.* use two different agents for their generic game level generator. Their approach maintains two populations, feasible and infeasible, of candidates and tests them with a *OneStepLookAhead* and *DoNothing* agents. The performance of the *OneStepLookAhead* agent is used to evaluate the feasible population while the *DoNothing* agent checks for player death within a certain timestep. Simulation is not limited to level creation; Gravina *et al.* in [47] experimented with generating weapons in Unreal Tournament 3. After a simulated game between agents fitness was calculated based on kill distribution. A user study showed the evolved weapons were percieved as both balanced and fun.

Evaluating generated content via simulation can be an effective approach at the cost of far higher computational and implementation complexity. Though faster than real time testing, simulations are still very slow relative to a direct fitness algorithm thus making online generation unlikely. Similarly, as discussed in [41], an agent to carry out the simulation may not be readily available requiring time and resources to be invested into developing one; undermining one of the core reasons for a developer to use PCG.

The final approach is to algorithmically evaluate the fitness of content, this is known as Direct Evaluation. It involves extracting out certain characteristics, such as number of paths, resource density etc, of the candidate and pass them directly to the function. Ferreira *et al.* demonstrate direct evaluation in their Super Mario Bros. level generator.[45] They maintain four evolution populations of arrays representing the ground, blocks, coins and enemies of the level. These are all evolved separately and fitness is calculated as the difference between the individual and target sparseness (entropy for ground). While this provides a very quick evaluation of individuals resulting in a much faster process describing the level structure in terms of entropy is not very intuitive for designers.Ashlock *et al.* formalise direct fitness functions for generating mazes in [46]. By defining the entry and exit points along with a number of checkpoints several fitness functions can be defined that reward different maze features. A global constraint, *k,* is defined to specify the number of checkpoints that lie on a path to the exit. The functions outlined:

1. Exit Path Length: Length of the path from start to end; generally created single winding paths.
2. Primary Reconvergence Sum: A sum of the shortest path between the start point and pairs of checkpoints; encourages all checkpoints to be accessible.
3. Isolated Primary Reconvergence Sum: A sum of the shortest path between the start point and pairs of checkpoints which only passes through those checkpoints and no others; rewards branching paths to each checkpoint before reconverging.
4. Cul-de-sac Count: Number of dead end paths. Combined with the constraint that all checkpoints are accessible rewards looping paths.
5. Cul-de-sac Length: Sum of the distance between entrance and dead ends. Maximises the path length from entrance to exit but creates more side paths than 1.

The above functions were tested on four distinct maze representations and produced easily comparable results suggesting high genericity and thus suitable for reuse in multiple projects.

The quality of game content often cannot easily be described by a single function as there are naturally a number of aspects that contribute. Togelius *et al.* experimented with using a multi-objective algorithm to evolve complete StarCraft maps[43]. Here a number of fitness functions were tested to satisfy four main goals:

1. Playability: support normal gameplay
2. Fairness: all players have equal chance of winning.
3. Skill Differentiation: support multiple play styles.
4. Interestingness: avoid symmetrical or repeating elements.

The individual functions that optimise for specific aspects such as base distances, resource placement and pathing. 8 functions were tested in pairs to investigate conflicting optimisation goals. Unlike Single-Objective evolution that optimises multiple variables, Multi-objective seeks to balance the trade-offs between a number of conflicting objectives e.g. maximising car performance and minimising fuel consumption. This results in a potentially infinite number of Pareto Optimal solutions where none of the objectives can be improved without degrading the others. This means that the design of the fitness functions and which of the Pareto solutions that are selected can have a significant impact on the quality of the output. In other words, optimising for two completely conflicting goals could potentially produce content that poorly satisfies both.

Direct evaluation of fitness is generally the least gameplay intrusive and fastest method, though efficiency can vary wildly between implementations, allowing for its use at runtime. However designing functions that can quickly and accurately quantify fitness is a complex challenge, especially when optimising multiple objectives.

2.4 Narrative

This section will discuses the procedural generation of narrative within games. A relatively unexplored area of research, narrative generation is less formally defined than the previously discussed approaches. Similarly, due to the unique challenges of narrative generation, the constructive and search based techniques do not easily translate across thus it will be discussed as a related but separate problem.

To further quantify narrative generation there is a distinction between what this paper refers to as Intentional Narrative (IN) and Emergent Narrative (EN). IN is a story or quest line, whether randomly generated or explicitly scripted, the designer has intended for the player to experience. While EN is the process of the player creating their own narrative based on the interaction of elements within a rule-based game world. [48] In other words, both IN and EN can be generated narratives but the former uses a specific game system for this purpose while EN uses the game world to support the player's imagination.

The systematic nature of games that make heavy use of PCG naturally lend themselves to EN generation [49]. [16][50][52] are a number of successful commercial games that make use of PCG and EN. However, PCG is not a required for games to make use of EN. Facade[58] creates an emergent plot through the player's interaction with two AI agents and objects within their house. Similarly, the hugely popular EVE Online[52] has generated a number of complex narratives just through player interaction [53-55] and has even had a novel written based on them.[56]

The issue with EN is that, while each narrative is unique and player-driven, it relies entirely on the player to find meaning behind the interactions of the game world. Less imaginative players or those who do not witness a novel interaction of game elements can miss out entirely on any narrative. Similarly designers have little control over what the player experiences.

The generation of an interesting, cohesive narrative for players is significantly complex. Implementation of IN generation in commercial games is largely simplistic optional content such as the “Radiant” quest system in Skyrim.[57] This can result in predictable and repetitive content with a systematic feel.

One of the earliest examples of an IN generation system was TALE-SPIN[60] which would simulate a plotline using pre-defined characters and world. However, there is little control of the narrative beyond defining character goals.

Li *et al.* [61] demonstrate generating a game narrative tailored to player preferences by adapting a pre-written plot. By using partial order planning they attempt to optimise the plot structure according to the player model while maintaining coherence and the original author's intent. As touched upon in previous sections, players generally prefer content adapted to their preferences and the testing Li *et al.* carried out shows the same applies for game narratives. However, the requirement of an entire pre-existing game plot to serve as a basis is a significant drawback.

In [62] Kybartas *et al.* design a promising system for the generation of complete quests that reflect and impact the current state of the world in Role Playing Games(RPGs). The system represents the game world as a directed multi-graph with game entities (NPCs, items etc) as the nodes and relations as edges. Narratives are represented as a directed acyclic graph of events. Quest generation uses designer written grammar-rules which search for a specific relation within the world graph as the antecedent and a simple narrative graph as the consequence. For example, two NPCs who hate each other may generate an assassination quest. This initial narrative graph is then rewritten using a separate set of grammar-rules to expand and branch the questline. Finally the system simulates completion of the quest and updates the world graph to reflect any consequences. Kybartas *et al's* design allows for a high degree of control for the developer while maintaining the ability to generate varied content. Additionally, the system respecting the player's actions and taking account of previous quests would help players feel their choices have a real effect on the game world. While Kybartas *et al.* do not provide details, maintaining the world graph would likely be computationally expensive and from a practical standpoint, the initial creation of the world graph for large scale games would be a time consuming process.

The generation of interesting and cohesive narratives has yet to be achieved in games beyond limited experimental games but with the industry's continued focus on PCG it is likely more games using some form of narrative generation will be developed.

3. Conclusion

The choice of PCG technique will have a huge impact on the performance, diversity of output and complexity of the system. With the overwhelming range of approaches available it is crucial to consider the both the content and the context it will be generated in.

The choice to base my system on the android platform places relatively strict restrictions on the algorithm's complexity and resource usage due to the more limited hardware of mobile devices. In deference to these constraints the base terrain elevation will be generated using noise while the placement of game objects will be handled by a genetic algorithm. Noise was chosen as functions to evaluate the fitness of terrain are often computationally expensive and can produce unnatural looking levels. The loss of control over the generated terrain features is an acceptable trade off for the performance gain. It has been demonstrated that purely random levels are less well received by players than more directed generation. The placement of game objects by the genetic algorithm will help offset the randomness of the noise terrain. This approach takes advantage of both the speed of constructive generation and quality of search based.

**References**

1. A. Travers. "GTA 5' Costs $265 Million To Develop And Market, Making It The Most Expensive Video Game Ever Produced: Report". February 2013. GamesIndustry.biz, [Online] Available : <http://www.ibtimes.com/gta-5-costs-265-million-develop-market-making-it-most-expensive-video-game-ever-produced-report> Retrieved: 28/10/16
2. A. Drachen, A. Canossa, and G. N. Yannakakis, “Player modeling using self-organization in tomb raider: Underworld.”. September, 2009. Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Milano, Italy.
3. B. Kybartas, C. Verbrugge. “Analysis of ReGEN as a Graph-Rewriting System for Quest Generation”. November 2013. IEEE Transactions on Computational Intelligence and AI in Games (Volume: 6, Issue: 2)
4. J. Togelius, G. N. Yannakakis, K. O. Stanley, C. Browne. “Search-Based Procedural Content Generation: A Taxonomy and Survey”. September 2011. IEEE Transactions on Computational Intelligence and AI in Games (Volume: 3, Issue: 3)
5. T. Warren. "Minecraft sales top 100 million". June 2016. The Verge. [Online] Available: <http://www.theverge.com/2016/6/2/11838036/minecraft-sales-100-million> Retrieved 28/10/16.
6. Elite, Acornsoft. 1984.
7. .kkrieger, Farbrausch. 2004.
8. SpeedTree. Interactive Data Visualization, Inc. [Online] Available: <http://www.speedtree.com/>
9. Fournier, Alain; Fussell, Don; Carpenter, Loren. 1982. "Computer rendering of stochastic models". Communications of the ACM.
10. Miller, G.S.P. “The definition and rendering of terrain maps”. 1986. Proceedings of the 13th annual conference on Computer graphics and interactive techniques. SIGGRAPH
11. Lewis, J. P. “Generalized stochastic subdivision”. 1987 ACM Transactions on Graphics.
12. Perlin, Ken. “An Image Synthesizer”. 1985. Proceedings of the 12th annual conference on Computer graphics and interactive techniques. SIGGRAPH
13. Ken Perlin. “Noise hardware”. 2001. Real-Time Shading SIGGRAPH Course Notes.
14. Ken Perlin. “Improving Noise”2002. Proceedings of the 29th annual conference on Computer graphics and interactive techniques. SIGGRAPH
15. Kurt Spencer. “Noise!”. [Online] Available : <http://uniblock.tumblr.com/> Retrieved 04/11/16
16. Minecraft. Mojang. 2011
17. Markus Persson. "Terrain generation, Part 1” March 2011. [Online] Available: <http://notch.tumblr.com/post/3746989361/terrain-generation-part-1> Retrieved: 04/11/16
18. Travis Archer. “Procedurally Generating Terrain” 2011. Morningside College.
19. Howard Zhou, Jie Sun, Greg Turk, James M. Rehg. “Terrain Synthesis from Digital Elevation Models”. 2007. IEEE Transactions On Visualization and Computer Graphics. Vol.13. No.4
20. Jonathon Doran, Ian Parberry “Controlled Procedural Terrain Generation Using Software Agents”. 2010. IEEE Transactions On Computational Intelligence and AI in Games Vol.2 No. 2
21. LEWIS, J. P. 1990. Is the Fractal Model Appropriate for Terrain? Disney Secret Lab.
22. F. Kenton Musgrave, Craig E. Kolb and Robert S. Mace. “The Synthesis and Rendering of Eroded Fractal Terrains.” 1989. In Proceedings of the 16th annual conference on Computer graphics and interactive techniques.
23. Kenji Nagashima “Computer generation of eroded valley and mountain terrains.” 1997. The Visual Computer, p.g.456-464.
24. D. D’Ambrosio, S. Di Gregorio, S. Gabriele, R. Gaudio “A Cellular Automata Model for Soil Erosion by Water.”. Department of Mathematics, University of Calabria.
25. N. Chiba, K. Muraoka and K. Fujita “An Erosion Model Based on Velocity Fields for the Visual Simulation of Mountain Scenery.” 1998. The Journal of Visualization and Computer Animation, Vol.9
26. Bedrich Benes. “Visual simulation of hydraulic erosion”. 2002. Journal of WSCG 10.
27. Dwarf Fortress. Bay 12 Games. 2006
28. “World Generation” Dwarf Fortress Wiki.[Online] Available: <http://dwarffortresswiki.org/index.php/DF2014:World_generation> Retrieved:09/11/16
29. David E. Goldberg. “Genetic Algorithms in Search, Optimization, and Machine Learning”. 1989. ISBN: 0201157675
30. Antonios Liapis, Héctor P. Martínez, Julian Togelius, Georgios N. Yannakakis. “Adaptive game level creation through rank-based interactive evolution”. 2013. IEEE Conference on Computational Intelligence in Games.
31. William L. Raffe, Fabio Zambetta, Xiaodong Li, Kenneth O. Stanley. “Integrated Approach to Personalized Procedural Map Generation Using Evolutionary Algorithms”. 2014. IEEE Transactions on Computational Intelligence and AI in Games Vol.7 No.2
32. Kenneth O. Stanley, Risto Miikkulainen. “Evolving Neural Networks Through Augmenting Topologies”. 2002. Evolutionary Computation Vol.10
33. E. J. Hastings, R. K. Guha, and K. O. Stanley, “Evolving content in the Galactic Arms Race video game,” 2009. IEEE Transactions on Computational Intelligence and AI in Games. Vol.1 No.4
34. Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall, Kenneth O. Stanley. “Combining Search-Based Procedural Content Generation and Social Gaming in the Petalz Video Game”. 2012. Eighth Artificial Intelligence and Interactive Digital Entertainment Conference.
35. David Stammer, Tobias Gunther, Mike Preuss.“Player-Adaptive Spelunky Level Generation”. 2015. IEEE Conference on Computational Intelligence in Games
36. Christopher Pedersen, Julian Togelius, Georgios N. Yannakakis. “Modeling Player Experience for Content Creation”. 2010. Transactions on Computational Intelligence and AI in Games Vol.2 No.1
37. Georgios N. Yannakakis, Julian Togelius. “Experience-Driven Procedural Content Generation”. 2011. Transactions on Affective Computing.
38. Nevermind. Flying Mollusk. 2015.
39. Tropical Heat: Jet Ski Racing. SomaticVision.
40. Jordan Fisher. “How to Make Insane, Procedural Platformer Levels”. 2012. Gamasutra. [Online]. Available: <http://www.gamasutra.com/view/feature/170049/how_to_make_insane_procedural_.php> Retrieved: 10/11/16
41. Mohammad Shaker, Noor Shaker, Julian Togelius. “Evolving Playable Content for Cut the Rope through a Simulation-Based Approach”. 2013. Faculty of Information Technology Engineering, Damascus, Syria.
42. Mohammad Shaker, Noor Shaker, Julian Togelius, Mohamed Abou-Zleikha. “A Progressive Approach to Content Generation”. 2015. Joseph Fourier University.
43. Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelback, Georgios N. Yannakakis. “Multiobjective Exploration of the StarCraft Map Space”. 2010. IEEE Conference on Computational Intelligence and Games.
44. Ahmed Khalifa, Diego Perez-Liebana, Simon M. Lucas, Julian Togelius. “General Video Game Level Generation”. 2016. Proceedings of the Genetic and Evolutionary Computation Conference 2016.
45. Lucas Ferreira, Leonardo Pereira, Claudio Toledo. “A Multi-population Genetic Algorithm for Procedural Generation of Levels for Platform Games.”. 2014. Proceedings of the Genetic and Evolutionary Computation Conference 2014.
46. Daniel Ashlock, Colin Lee, Cameron McGuinness. “Search-Based Procedural Generation of Maze-Like Levels”. 2011. IEEE Transactions On Computational Intelligence and AI in Games.
47. Daniele Gravina, Daniele Loiacono. “Procedural Weapons Generation for Unreal Tournament III”. 2015. IEEE Games Entertainment Media Conference
48. P. Sweetser and J. Wiles, “Scripting versus emergence: Issues for game developers and players in game environment design,” 2006. International Journal of Intelligent Games and Simulation, Vol.4 No.1
49. Simon Chauvin, Guillaume Levieux, Jean-Yves Donnar, Stephane Natkin.“Making Sense of Emergent Narratives: An Architecture Supporting Player-Triggered Narrative Processes”. 2015. IEEE Conference on Computational Intelligence in Games.
50. “Terraria,” Re-Logic, 2011.
51. “Don't Starve”. K-Entertainment. 2013.
52. “EVE Online”. CCP Games. 2003.
53. “Forget to pay your virtual game bill? Could cost you $200,000”. CBS News. [Online]. Available: <http://www.cbsnews.com/news/eve-online-mmorpg-battle-worth-200000/> Retrieved: 16/11/16.
54. Brendan Caldwell. “A War In Space: The Banker Who Gambled Everything And Brought Eve’s Greatest Empire To Its Knees”. Rock Paper Shotgun. [Online]. Available: <https://www.rockpapershotgun.com/2016/04/21/eve-online-world-war-bee-mittani/> Retrieved: 16/11/16.
55. Bo Moore. “Inside the Epic Online Space Battle That Cost Gamers $300,000”. Wired. [Online]. Available: <https://www.wired.com/2014/02/eve-online-battle-of-b-r/> Retrieved: 16/11/16.
56. Andrew Groen. “Empires of EVE: A History of the Great Wars of EVE Online”. 2016. ISBN: 9780990972426
57. J. Horsey, “The Elder Scrolls V: Skyrim radiant quest system, will customise your game as you play,” Geeky Gadgets, Nov. 2011 [Online]. Available: <http://www.geeky-gadgets.com/the-elder-scrolls-v-skyrimradiant-quest-system-will-cusomtise-your-game-as-you-play-10-11-2011/> Retrieved: 16/11/16
58. “Facade”. Michael Mateas, Andrew Stern. 2005.
59. “Event[0]”. Ocelot Society. 2016.
60. J. R. Meehan, “TALE-SPIN, an interactive program that writes stories,” 1977. In Proceedings of 5th International Joint Conference on Artificial Intelligence. San Francisco, CA, USA, 1977
61. Boyang Li, Mark O. Riedl. “An Offline Planning Approach to Game Plotline Adaptation”. 2010. Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.
62. Ben Kybartas, Clark Verbrugge. “Analysis of ReGEN as a Graph-Rewriting System for Quest Generation”. 2014. IEEE Transactions On Computational Intelligence and AI in Games. Vol.6 No.2
63. “Euphoria”. NaturalMotion. 2015. [Online] Available: <http://www.naturalmotion.com/middleware/euphoria> Retrieved: 20/11/16

1. AAA (pronounced triple A) is a classification of games with the highest development and marketing budgets and usually form the years bestsellers. [↑](#footnote-ref-2)