Would You Look at That! Vision-Driven Procedural Level Design

Michael Cook
AIR Lab, University of Falmouth
mike@gamesbyangelina.org

Abstract

In this paper we present a technique for procedurally generating sections of 3D level geometry using computational evolution and guided by the visibility of certain game objects or areas during play. We show that certain level design goals can be achieved in the resulting levels, such as encouraging or dissuading player sightings of certain objects or locations. We also give details of a simple study of players on the generated levels, and discuss how this might be expanded to incorporate more complex problems related to level design.

Introduction

Designing 3D levels for games is an extremely complex and multi-faceted task. It often requires an understanding and appreciation for many different disciplines, including architecture, art direction, colour theory, psychology, lighting, and camera direction. Levels must often be exquisitely designed to achieve certain aesthetic goals, but they must also balance this with many functional goals, too, such as providing spaces for the player to engage with a game's systems, or ensuring content is paced and ordered correctly.

Another important function of level design is to draw the player's attention to certain areas, in order to lead their exploration, to teach them about a game mechanic, or simply to reinforce an element of the game narrative. For example, in the first-person narrative game *Dear Esther*, a plot-significant radio tower is prominently visible for much of the game, deliberately and specifically framed by the landscape and the player's path through the island, before the player is made aware of its significance. In *Mirror's Edge* the level design uses colour coding and architectural structure to guide the player towards the exit to a level or play area – an example of which is shown in Figure 1. Good level design can suggest or encourage a player without explicitly forcing them to perform actions or taking away control, which is a much more preferable way of conveying information as it does not disrupt immersion, flow or the player's sense that they are in control of the game.

Procedural content generation is a rapidly expanding area both within commercial game development and academic research. Some generative approaches and content types are deeply iterated upon in order to find better and more reliable ways to do them, such as dungeon generation (Horswill and Foged 2012), while other work focuses on breadth and finding new techniques and new kinds of content to generate, such as discovering new classes for RPGs (Pantaleev 2012). At the same time, the emerging field of automated game design seeks to integrate procedural generation research at a higher level, providing software with the capacity to make design decisions and express them through generative processes.

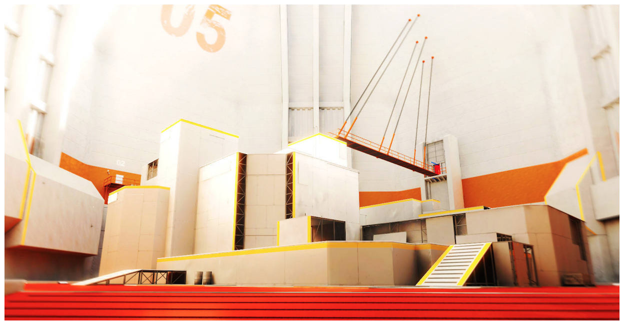
Figure 1: A room in *Mirror's Edge*. Colour and the clear lines of the suspended walkway draw the player's eye towards the exit door.

ANGELINA is a piece of software which automatically designs games, built using ideas from procedural content generation as well as computational creativity (Cook and Colton 2011). The system has designed games in many genres, including simple 2D arcade games and puzzle-platformers. ANGELINA has primarily focused on designing games with reflex-based challenges or skill checks of some kind, something which is a common theme throughout automated game design research currently (Cook and Smith 2015). We have previously argued that automated game design must strive to be a broad and all-encompassing discipline that understands the many facets and interleaved elements of game design. This paper explores an existing area of procedural generation research – level generation – with a new angle, focusing on how the vision of the player can be incorporated to achieve both functional and aesthetic design goals. This represents another step towards a new version of ANGELINA, which we hope will be able to demonstrate a richer and more nuanced understanding of how 3D spaces are designed, by being able to incorporate ideas such as what the player sees into its games.

In this paper we present a system for generating the designs of 3D spaces by placing geometry in a world that the player must navigate through. The generation process is guided by how much or how little of certain markers the player sees during their path through the world. We discuss how the limitations of our current approach could be solved in future iterations of the system, and report on a short pilot study which suggests the system works in its current, simple form.

Related Work In 3D Level Design

A majority of work on procedural level design applies to two-dimensional levels, in particular side-on platformers in the style of *Super Mario* such as (Shaker et al. 2011). Work by Cardamone et al. in (Cardamone et al. 2011) looked at generating 3D levels for competitive scenarios in multiplayer games, with a focus on how the design of the level influenced combat interactions between players. This has since been extended to look at level balance in (Lanzi, Loiacono, and Stucchi 2014). There has also been interested in level design for single-player experiences, such as (Lopes, Liapis, and Yannakakis 2015) which looks at atmospheric design for 3D horror game levels, combining audio placement with level geometry.

Nitsche et al. have explicitly explored the relationship between procedural generation, level design and player experience (Nitsche et al. 2006). The focus here is on empowering the player to have control or agency in the generative process, which is generally not our focus here: instead, we are more interested in offline generative processes which occur before the game is finished and presented to the player. However, the authors ask very important questions about the relationship between procedural generation and authored experiences. In particular, their discussion about conflict between generated and designed spaces closely parallels the motivations for our work on ANGELINA:

> *The mere fact, that we can generate space does not mean that this space necessarily makes any sense or is of any value to the player. Procedurally generated game worlds can stretch into infinity but the meaning of each single locale can be thinned out by that.*

The authors' system *Charbitat* overcomes this partly by relying on hand-authored content which is reused by the generative system within procedural spaces. With ANGELINA we hope to build a system capable of providing its own sense of detail and design, without relying on people for large chunks of content.

In this paper we consider related but separate problems to the above work: we are interested in how the objects visible to a player during the game can be used to guide the design and layout of level geometry, with a focus on single-player experiences. This may have applications to multiplayer games as well, since level readability is very important in competitive situations (Gaynor 2009).

System Description

In this section we describe our system for generating level designs, and describe how the system is augmented to track the visibility of objects, as well as detailing the fitness functions used to evaluate the levels. In the following section we describe specific experiments using this system to illustrate its potential output.

Basic Level Generation

For the purposes of this paper, a level is a square area of game space, surrounded by walls the player cannot climb over. The player begins at a fixed location and must reach a defined 'exit object' to complete the level. Clearly this doesn't represent any particular game, but instead models a recurring subtask within games, namely moving from one location in a 3D space to another. A level design is represented as a collection of cuboid objects which are placed in the game world and obstruct vision and movement. We generate the level designs in this paper using computational evolution.

The population of level designs is initialised by randomly creating a number of cuboids (the exact number is randomised within a set minimum and maximum) with randomly-set positions and size (referred to within Unity as a three-dimensional vector called *scale*) within minimum and maximum levels. The parameters used for the levels shown in this paper are all defined in table 1.

Crossover of two level designs is performed as one-point crossover on the list of cuboids making up the level. Mutation occurs at a rate of 5% and randomly replaces a cuboid with a random cuboid, or randomly varies either the position or scale of the cuboid, while obeying the maximum and minimum constraints on these values.

For basic level generation, the system's objective is simply to create paths that cause the player to take longer paths through the game space in order to reach the exit. Therefore to evaluate a level design, the system takes control of the player object, plots a path to the exit using A*, and records how long it takes for the player to move there. Fitness is simply the time taken to reach the exit, with a negative fitness returned if the player gets stuck and can't progress or the path takes an abnormally long time (greater than thirty seconds for our examples here). This results in increasingly long and convoluted paths to the exit. Figure 2 shows a top-down view of a level evolved with a population of 30 level designs over 10 generations.

Parameter	Min	Max
Number of Cubes	25	35
Width	1	8
Height	0.5	8
Depth	1	8

Table 1: Parameters used to generate levels shown in this paper. Scale is in Unity base units.

Adding Vision

In order to add an understanding of vision to the system, we attached a 'camera' to the player object being moved through the levels. Objects tagged as 'vision markers' were introduced to the level. When moving through the level, the camera checks to see if any vision markers are within the frustrum of the camera. This tells us if the object is within the camera's bounds, but does not tell us if it is obscured by other object such as level geometry. To confirm visibility, the camera then calculates the vertices of the model and casts a ray to each vertex from the camera. For the examples described in this paper, we consider an object visible if one or more of these raycasts are successful. This could be made more strict, for example requiring that half or even all of the raycasts succeed, to make the object's visibility more explicit.

Once an object becomes visible, the camera tracks the duration of its visibility and stores the duration once the object stops being visible again. It can track multiple objects distinctly, meaning that the visibility of different objects can be combined and distinguished between in the fitness functions. For the examples in this paper we use a variety of fitness functions, however they generally involve a linear combination of the sum of the visibility intervals for each marker, with an optional additional parameter capturing the length of the path to the exit. We describe the fitness functions in more detail as we describe each experiment in our results section.

Results

In this section we show several results obtained with different fitness functions and vision marker setups. These results are shown without evaluation, and without curation, as an example of what the system is capable of producing. We then took several outputs from the system and ran a pilot study with people who played the levels and reported on object visibility. This is partly to compare visibility readings, but also shines a light on the importance of the player modelling used during evolution, which we discuss in subsequent sections.

Illustrative Results

All Markers Visible Figure 3 shows the resulting level from a run of the system with two hand-placed visibility markers, one red and one green, and the condition that both should be visible as much as possible on the path to the exit. The fitness function used is the mean visibility of all markers, defined as:

$$dmean(M) = \frac{\sum\limits_{m \in M} d(m)}{|M|} \tag{1}$$

Where M is a set of visibility markers, all of which we wish to be visible to the player, and $d(m)$ is the duration in seconds that m was visible for $m \in M$. The result is a path where the player takes a long diagonal across the level, passing by the red marker and looking directly at the green marker as they travel through the main level area. Note the fitness function does not include the time spent travelling to the exit, since an increased time spent looking at the objects implies that a longer time was spent travelling in general.

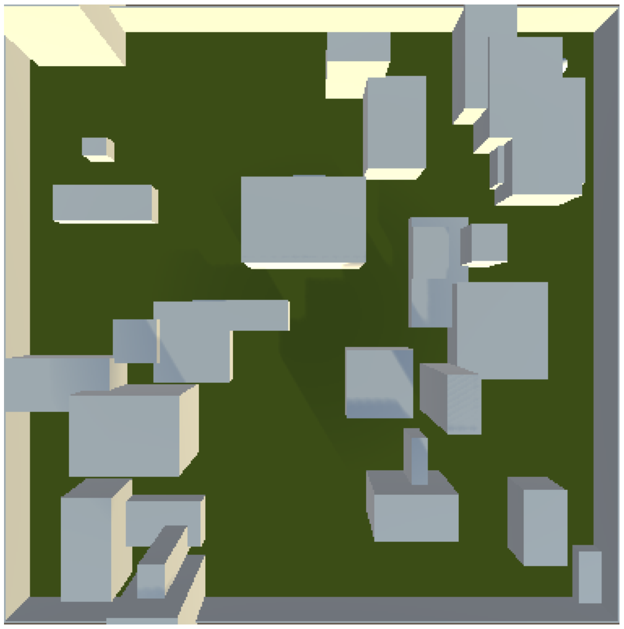
Figure 2: A sample level generated by the system, without visibility constraints. The player starts at the bottom-left corner of the image and must reach the top-right corner.

Some Markers Visible Figure 4 shows the level resulting from a run of the system with two hand-placed visibility markers as with the previous example. On this occasion, the red marker's visibilty should be *minimised*, while the green marker's visibility should be maximised. The fitness function we use is defined as follows:

$$f(M_{max}, M_{min}) = dmean(M_{max}) - dmean(M_{min})$$

Where for any given set of visibility markers M, $dmean(M)$ is defined as in (1), M_{max} is a set of markers whose visibility we wish to maximise, and M_{min} is a set of markers whose visibility we wish to minimise. As before, we do not include the overall time spent travelling as this is incorporated implicitly into maximising the visibility of certain markers. The resulting path leads the player away from the red marker, into an open area where the green marker is clearly visible on the way to the exit.

No Markers Visible Figure 5 shows a level evolved with two hand-placed visibility markers as before. This time both visibility markers should be unseen as the player paths to the exit. In this case, we add in the path length to the fitness function in order to encourage longer paths (otherwise the system simply converges on straight-line paths that don't see either marker). However, we weight the fitness loss from seeing markers higher than the path length, otherwise the system can evolve paths that run past the markers quickly and then take very long paths to rebalance fitness. By penalising marker sight more heavily, the system is encouraged

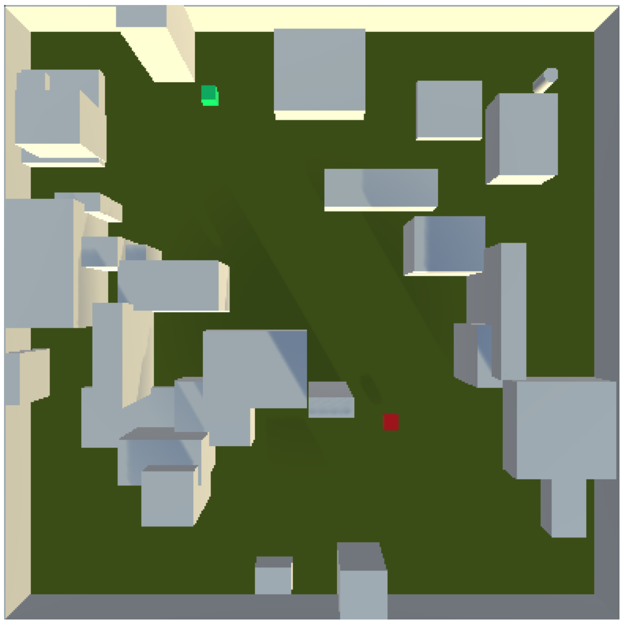
Figure 3: An evolved level design, constrained by both hand-placed Red and Green markers being visible on the path. The player starts at the bottom-left corner of the image and must reach the top-right corner.

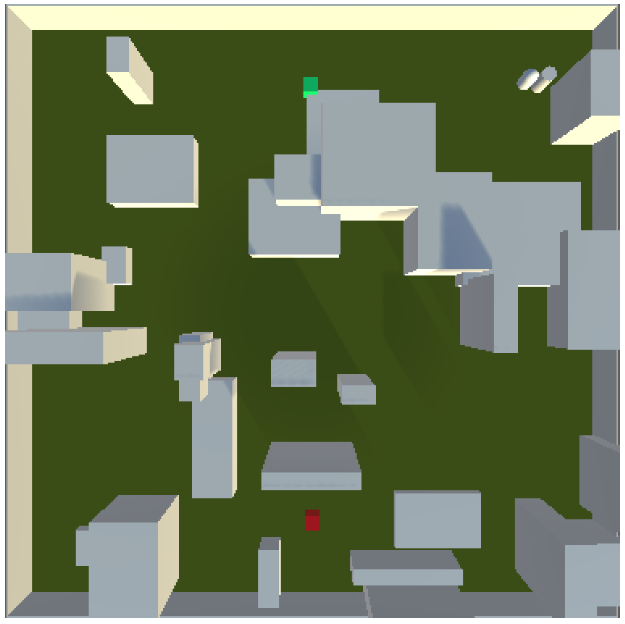
Figure 4: An evolved level design, constrained by the Green marker being visible, and the Red marker not being visible. The player starts at the bottom-left corner of the image and must reach the top-right corner.

to find longer paths that avoid them altogether. The fitness function:

$$f(M, PL) = PL - dmean(M)$$

Where $dmean(M)$ is defined as in (1), M is the set of markers, all of which we wish to avoid seeing, and PL is the length of the path in seconds. We include the path length here since the duration a marker is visible for cannot be negative, which produces a flat fitness gradient. The evolution does work without path length being incorporated, but the resulting levels were considered less interesting in experimentation. The result in Figure 5 shows two approaches to avoiding vision of markers: in the case of the green marker it's simply placed behind blind corners that the player doesn't see on a straight-line path to the exit. However, the red marker has been placed inside a piece of geometry (we exaggerate its height in the figure to make it visible). These are two very different ways of fulfilling the fitness function, and we discuss this in the following sections.

Pilot Study

As we discuss in the next section, the current pathfinding methods and the task of seeking a goal do not necessarily accurately represent how players navigate through 3D worlds. Nevertheless, we conducted a simple study using three levels generated by the system to investigate how successful the evolution is in hiding or showing level sections to the player. We generated three levels using the same settings as those in the results shown above: one with both markers not visible, one with one marker visible and the other not, and one with both markers visible.

We asked players to find the exit in each level (we facilitated this by making the exit project a column of light upwards into the sky so the players could gauge the rough bearing of the goal). After completing each level, players were asked to indicate if they noticed any coloured markers in the level. We offered four possibilities: red, green, purple and orange. We also asked players to indicate if they gave up trying to reach the exit (since this task is quite difficult and it is worth noting if the system generated a path which was obscure or hard to find).

The study had ten participants in total, nine of whom are people with long-term histories of playing games. All ten participants completed all three levels and found the exit, and on all three levels all participants correctly identified the markers that were intended to be visible, did not identify markers that should not be visible, and did not raise false positives by identifying markers that were not in the level (such as the orange marker). In other words, the levels performed exactly to specification in all three cases.

Although the data matches up well with the evolved level specifications, we consider this only a cursory indicator of the system's quality - these levels are currently relatively simple to design, as we discuss in the following section, and so it is not hugely surprising perhaps that the system performs well. However we are nevertheless pleased that it confirms the basic ideas behind the system and hope that we can follow this up with more substantial surveys on this work as the system is developed further. Validating a model of player curiosity, as we discuss below, should prove a more difficult task.

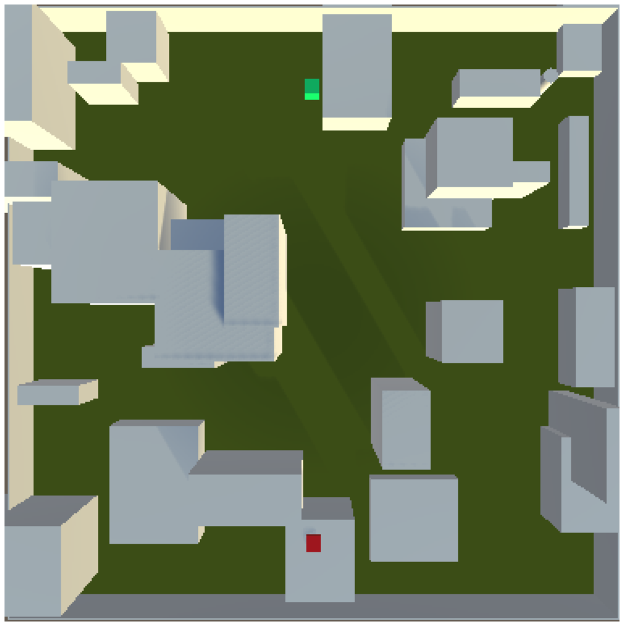

Figure 5: An evolved level design, constrained by both the Green and Red markers not being visible. The player starts at the bottom-left corner of the image and must reach the top-right corner. The Red marker height has been exaggerated so its position inside the geometry is visible.

Evaluation and Discussion

A* Pathfinding Accuracy

Naturally, the AI pathfinding processes used by the simulation do not accurately represent how players will move through a 3D space. We use such pathfinding here primarily as proof-of-concept of the system and as a useful starting point to explore some of the ideas of the project. Pathfinding is imperfect for two main reasons: in games with explicit pathfinding goals (such as reaching an exit) the player must still explore and understand the space to find the exit. Other games allow or encourage exploration at leisure, and are thus less directed than the pathfinding we use. A major point of future work which we hope to explore next is building models of player movement beyond point-to-point pathfinding that incorporate ideas such as visual curiosity and exploration. Despite this weakness we're happy with the results from this system, and we feel that even with this basic form of simulation it shows that the approach can carve out meaningful level design sketches.

Obscurity vs. Hiding

We don't define what the vision markers represent in this system. As we saw in the results, the evolutionary system sometimes hides markers by placing them inside geometry, and sometimes hides them in open space but behind obscuring objects. In the former case, markers represent areas which the player should be zoned out from somehow - like an inverse reachability marker. They indicate constraints for the paths produced by the level designer. In the latter case, markers represent areas which are off the critical path or made secret from the player somehow, but are still theoretically accessible. These solve very different design problems - we believe that we can distinguish between these two cases in future versions of the system by implementing secondary pathfinding objects that seek out the vision markers and decide whether or not they can be accessed regardless of their visibility.

3D Model Usage

In order for this to apply to real-world level design problems, the use of cube primitives as level geometry would need replacing with a database of complex 3D models. Using non-primitive 3D models complicates the generation problem with our current approach - cubes can be enlarged to any size or shape without looking unusual, which is not true of most 3D models. Additionally, most game models will have more complex silhouettes and this might mean it's harder to find high-fitness results as obscuring the player's vision may be more difficult. However, this only emphasises the interestingness of the problem from a computational standpoint – we're eager to see what results can be obtained on this harder problem.

One way to overcome this might be to use a database with a variety of 3D models and selecting from these models rather than attempting to adjust scale. In doing so we reduce the generative space but allow for custom model sets to be used in level generation. As long as these models are diverse in their size and shape, it should also be possible to find placements that obscure vision enough (such as large buildings). This would enable the tool to be more useful in an applied setting, as it can work with real game content and understand constraints on their placement to produce more useful, more realistic level sketches. We don't consider it an urgent point of future work, but it is definitely a valuable line of inquiry. Even in its current state, we might consider the generator similar to the practice of *greyboxing* in level design, where cubes are used to sketch out the overall structure of models in a level.

Computational Level Design

Finally, it's important to point out that this work represents an incredibly simplified idea of level design, and then examines one aspect in excruciating detail. While we report on this detail in this paper, we do so with a view to extending this work in the future, and building a system which can understand more than simply where an object is placed and when it is seen. Even this task has many nuances not captured here: does the object stand out against its background? How is the object lit? Does the player know what the object signifies? There are many complex design problems all interlinked here, and we offer this only as one possible starting point into this area of rich and exciting questions to consider. This is a simplification, but hopefully a useful isolated example to study.

Conclusions

In this paper we presented an evolutionary system for designing 3D level layouts that are constrained by the objects the player sees on their path through the world. We show that visibility constraints can easily be engineered into such a system and that we can achieve level design goals such as encouraging player attention onto an object. We discussed the potential extensions for this work and its current limitations. 3D level design is an interesting and nuanced area of game development that we are only just beginning to explore with procedural generation – we believe this work sheds a little more light on another aspect of this interesting area.

Acknowledgements

The author wishes to thank Gillian Smith and Kaz Grace for informative discussion about game design, and Robert Yang who helped inspire this work. Thanks also to the reviewers of the paper who suggested many useful reading topics which will help the project grow.

References

Cardamone, L.; Yannakakis, G. N.; Togelius, J.; and Lanzi, P. L. 2011. Evolving interesting maps for a first person shooter. In *Applications of Evolutionary Computation*.

Cook, M., and Colton, S. 2011. Multi-faceted evolution of simple arcade games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*.

Cook, M., and Smith, G. 2015. Formalising non-formalism: Breaking the rules of automated game design. In *Proceedings of the Foundations of Digital Games Conference*.

Gaynor, S. 2009. Basics of effective fps encounter design. http://www.fullbrightdesign.com/2009/02/basics-of-effective-fps-encounter.html.

Horswill, I. D., and Foged, L. 2012. Fast procedural level population with playability constraints. In *Proceedings of the Artificial Intelligence in Interactive Digital Entertainment Conference*.

Lanzi, P. L.; Loiacono, D.; and Stucchi, R. 2014. Evolving maps for match balancing in first person shooters. In *IEEE Conference on Computational Intelligence and Games*.

Lopes, P.; Liapis, A.; and Yannakakis, G. N. 2015. Sonancia: Sonification of procedurally generated game levels. In *1st Computational Creativity and Games Workshop at the International Computational Creativity Conference*.

Nitsche, M.; Ashmore, C.; Hankinson, W.; Fitzpatrick, R.; Kelly, J.; and Margenau, K. 2006. Designing Procedural Game Spaces: A Case Study. In *FuturePlay*.

Pantaleev, A. 2012. In search of patterns: Disrupting rpg classes through procedural content generation. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games*.

Shaker, N.; Togelius, J.; Yannakakis, G. N.; Weber, B. G.; Shimizu, T.; Hashiyama, T.; Sorenson, N.; Pasquier, P.; Mawhorter, P. A.; Takahashi, G.; Smith, G.; and Baumgarten, R. 2011. The 2010 mario ai championship: Level generation track. *IEEE Transactions on Computational Intelligence and AI in Games* 3(4):332–347.