An Algorithmic Approach to Decorative Content Placement

Jonathan Tremblay
School of Computer Science
McGill University, Montréal
Québec, Canada
jtremblay@cs.mcgill.ca

Clark Verbrugge
School of Computer Science
McGill University, Montréal
Québec, Canada
clump@cs.mcgill.ca

Abstract

Placing decorative content in any virtual environment is not a trivial task, as the location of even non-critical content influences player movement choices through the level, as well as the general game aesthetic. In this work we use path visibility properties to define and investigate three methods for content placement in a game level, aimed at generating shared, unique, and never seen content. We compare and evaluate the three approaches using a human study where each player was given a level layout based on our three methods, showing our directed process has a non-trivial and controllable impact on player movement.

Introduction

Most digital games are goal-oriented; players are given an initial position and have to reach a certain goal position or state within a virtual level. Many generative methods to create such levels have been defined, and are able to create engaging levels (Dormans and Bakkes 2011), while making sure the game's fundamental puzzle structure in terms of keys, locks, chests *etc*., is kept or properly created. In order to create an interesting in-game experience, however, a level typically includes a variety of decorative content, ranging from textures to interactive objects. The placement of this content is still important to the player's experience in terms of inducing immersion and inspiring exploration, but is less well defined, and typically placed through manual designer effort, or in randomized locations within generative approaches.

In this paper we are interested in formalizing algorithmic processes for meaningful placement of decorative content. Our approach considers possible solutions (paths) through a level, computing a region of *weak visibility*, such that every point in that region is visible from somewhere on the path. We use that to determine what players may or may not encounter, and thus where content should be located so as to either be seen on all paths, be (relatively) unique to a path, or not typically be encountered on any common path. We validate our design through a human study demonstrating that player choices are materially affected by our placement strategies, and thus can server as a mechanism to encourage exploration or replay. Contributions of this work include:

- We define three different methods for positioning game content based on a discrete form of weak visibility regions.
- A human study is conducted, demonstrating that our placement approaches have a direct affect on player behaviour.

Background & Related Work

The study of visibility is one of the cornerstones of computational geometry as it arises naturally in computer graphics, robotics, digital games, *etc*. Many forms of visibility problems have been extensively studied in two or higher dimensions.

Given a polygon $\mathcal{P}$ of n vertices, the method to define a visibility polygon from a single point, q, is a well established problem (Ghosh 2007), of time complexity $\Theta(n \log(n))$. We use the well known angular plane-sweep algorithm (Asano 1985) to construct a visibility region $V(q)$, giving us a star-shaped polygonal region defined by the existing edge set, filtered according to visibility from q. Figure 2 shows such a region in light purple for point q.

In this work we are actually more interested in building the *weak visibility* polygon $V(s)$ of a segment s, where every point in $V(s)$ is visible from at least one point on s. Computing weak visibility has a surprisingly high theoretical complexity, and pathological cases can be constructed where $V(s)$ has $\Omega(n^4)$ vertices. Suri and ORourke gave a worst-case optimal algorithm, showed that weak visibility from a segment can be computed in $\mathcal{O}(n^4)$ time (Suri and O'Rourke 1986). To reduce the implementation complexity, however, we resort to a heuristic, discretized form of weak visibility, which we will present as part of our algorithmic methodology.

Related Work

We are generally interested in placing game content in a polygon such that it respects certain properties. The most well studied form of this problem can be expressed as placing single or multiple polygons *P* (content) inside a polygon *Q* while minimizing the Euclidean distance from the vertices of *P* to the vertices of *Q* (Aonuma et al. 1990). A problem more similar to the one of interest in this paper is the *chromatic art gallery* problem, aiming to ensure that landmarks

are distinguishable (Erickson and LaValle 2011). This is itself similar to the well known art gallery problem, but aiming at the minimum number of colours needed when guards are given different colours and their visibility regions overlap. This approach could be extended to apply to content positioning, although their technique does not take into consideration player movement within a level.

The research problem of generating high-level game-level maps is a well known problem; decorating a level with non-puzzle content, however, seems to be neglected. Dormans and Bakkes presented a grammar-based approach that generates missions that are mapped to a polygon representing a level (similar to the level we show in figure 9) (Dormans and Bakkes 2011), although the levels created are not populated with content. Smith *et al.* and Horswill & Foged used constraint solvers in order to assure that level puzzle structures were kept when populating a level with content (Smith et al. 2012; Horswill and Foged 2012).

Player motion is an important aspect of content decoration, as any choice taken by designer or in generative methods can influence the player's movement. Winters and Zhu presented a study that investigated the influence of level design paradigms on player path choices (Winters and Zhu 2014). They showed that shifted elevation such as stairs and directional lines had the most influence when making movement choices. Milam and El Nasr presented 5 design patterns used by developers to orient players based on their sight (Milam and El Nasr 2010).

Algorithmic Method

In this section we present the different algorithms used to produce unique and shared content in a level using visibility polygons. The method is composed of three major components: finding all path solutions, describing a path as a weak-visibility polygon, and finding content locations.

Finding Paths

The user of this technique provides a level, L, which we assume is composed of a 2D polygonal geometry (an overhead view). In figure 1, for example, the lighter area describes a polygon, with the central obstacle defining a single large hole. Within the level the user also provides starting, s, and goal, t, vector positions, as well as the desired number of shared, m, and unique, n, content locations.

In order to find all the possible paths from s to t, we proposed two approaches based on either using a roadmap and depth-first search, or using individual level traces[1] along with clustering.

Roadmap traces - We treat the problem of finding possible paths from s to t as equivalent to enumerating all simple paths from a corresponding node $v(s)$ to a corresponding node $v(t)$ in a graph G, generated from the roadmap. Note that this excludes cyclic routes, and so approximates direct path solutions rather than exploratory ones.

Arbitrary roadmaps may be used, as long as they result in a connected graph. In our design we construct a roadmap from a triangulation of the level geometry, building a graph G by associating the centres of triangles with nodes and adding edges between nodes if their triangles share an edge. Figure 1 shows such a triangulation and the roadmap retrieved from the dual-graph. For presentation purposes, the roadmap segments are drawn passing through the midpoint of each shared edge. In our implementation, the user can provide his own roadmap or use the presented technique to produce simple paths.

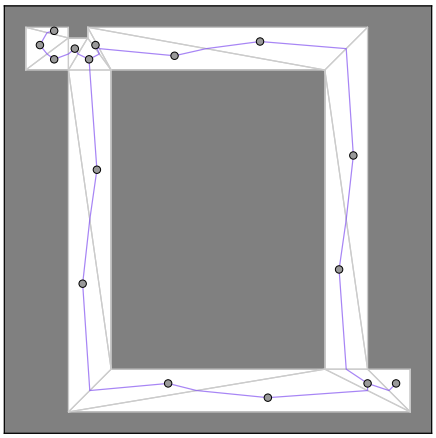


Figure 1: Level geometry with a triangulation-based roadmap (thin gray lines) and the graph representation built from its dual (gray circles and thin purple lines).

Given G, we can then proceed in finding all simple paths from s to t. As s and t are arbitrary 2 dimensional vectors, we begin by first locating the triangles enclosing s and t respectively, giving us appropriate starting and ending graph nodes $v(s)$ and $v(t)$. A recursive search is then conducted, using DFS to generate the set of all simple paths between $v(s)$ and $v(t)$. For games that allow for multiple entry and exit points this process can be repeated for all combinations of entry and exits. Although the problem of finding all simple paths between two (sets of) nodes has a worst-case exponential complexity, game roadmaps are relatively small and sparse (and here planar), and our experience is that a brute force approach is sufficiently fast at the scale of reasonable game-levels.

Clustering traces - In some situations possible player behaviours might be too complicated to be generated by a simple roadmap graph. Guard interactions in a stealth level, time-dependant actions, combat with enemies, *etc.,* tend to require players make careful choices in movement strategy, resulting in a much less exhaustive and less straightforward set of possible paths between start and goal than the underlying geometry may itself allow.

An alternative solution is then to build possible paths based on movement traces from actual gameplay. This could be achieved by recording human players as they go through the level, or mimicked by using a randomized game solver such as a rapidly exploring random tree to algorithmically generate a set of solutions (Tremblay, Torres, and Verbrugge 2014). Human and randomized traces tend to show significant fine-grain variability, so when adopting this approach

[1] produced by agent or human players

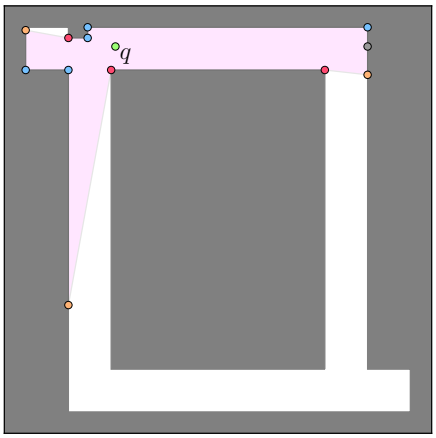


Figure 2: The visibility region (light pink) seen by vertex q (green vertex), the region is constructed from the apex vertices (red) and projections (orange), and the non-apex vertices (blue and black).

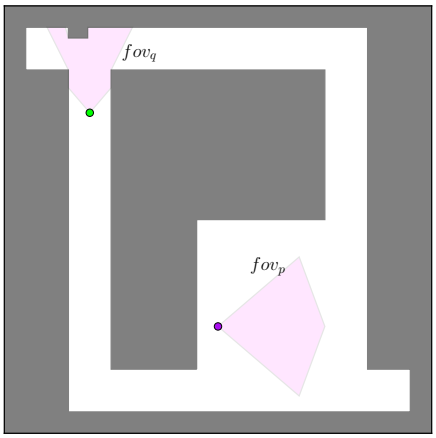


Figure 3: The light pink regions represent two FoVs, fov_p depicts the full, non-obstructed FoV of a point, whereas fov_q results from the intersection of a FoV with its visibility region.

we also rely on grouping similar traces together using clustering algorithms, following a geometric clustering approach by Campbell *et al.* (Campbell, Tremblay, and Verbrugge 2015). Once we have a set of path clusters, we can then construct representative paths from the centroid trace of each cluster.

Visibility Polygons

Using the collection of paths from s to t previously computed, we build a path weak-visibility region collection. This stage is separated into two steps: (1) constructing the visibility region from a single source, and (2) building the visibility polygon for the path.

Visibility region - Point visibility assumes a 360°, infinite range field of view (FoV). In most games, the player's ability to see is described by a more limited FoV, forming a cone of finite length and angle. This can be modelled in terms of straight-line polygons by a single triangle, or with more precision by multiple triangle strips. In our case we use two triangles as an efficient but still acceptable approximation, as shown in figure 3, as the pink area on the right. In order to construct the final visibility region we then intersect this polygon with the visibility region, giving us the final FoV region for a given point. The light pink region in the top left of figure 3 shows an example of the result.

Path visibility - Full path visibility implies a *weak* visibility calculation, computing the visibility region discernible from *any* point on the path. As previously discussed, weak visibility from even a single segment has high theoretical complexity. Our approach to path visibility is thus heuristic, based on combining a set of discrete, point-visibility computations, calculated at constant intervals along the path. This under-approximates actual weak visibility, but has a much lower implementation effort, and the accuracy versus cost trade-off can be easily controlled by altering interval size. For each point we build the point-visibility polygon, intersecting it with the player's FoV assuming they were looking towards the next point, or keeping orientation for the last point. The resulting FoV regions are merged for each point in the path, giving us the final path visibility.

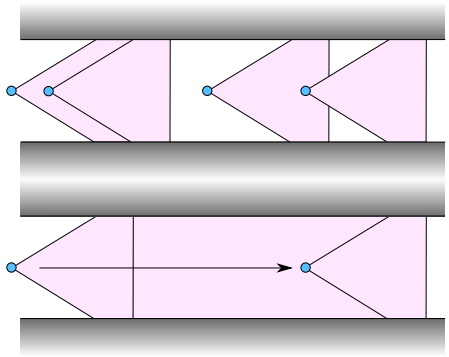


Figure 4: Computing weak visibility. A small interval size results in FoV polygons that would merge to fill the corridor (top corridor, left side), but with a larger step size can miss portions (top corridor, right size). The extruded FoV polygon (bottom corridor) is necessarily fully merged.

Naively done, and depending on the interval granularity, this calculation can result in a stepping effect, where portions of the actual visibility region are missing in our result. The top corridor of figure 4 shows two path visibility regions with different intervals. The movement on the left side uses a short interval, and the union of the two FoVs thus covers the full corridor. On the right side, a long interval is used, resulting in multiple missing triangles that actually would have been seen by the player walking along the segment.

Determining an optimal granularity is difficult—outside of simple corridor situations, ensuring sufficient overlap of individual player FoV polygons to avoid these missing areas can require arbitrarily small interval granularity. We notice, however, that the union of visibility polygons computed from the set of player positions does in fact include the missing areas, and they become excluded only when we intersect with each individual FoV. To avoid this problem, we thus

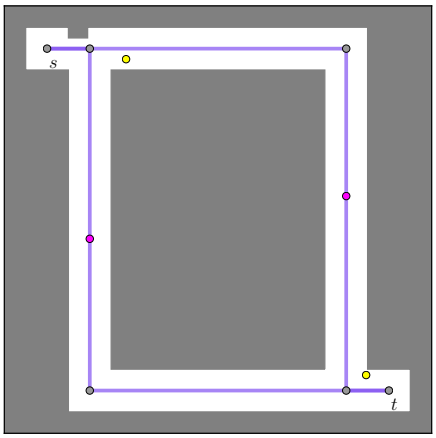


Figure 5: Final results including one shared content (yellow) and one unique content (red) per simple path from s to t.

compute the fully merged set of point-visibility polygons, intersecting it with a single polygon constructed by sweeping or extruding the player FoV from the starting position at the beginning of the segment to the ending position at the end of the segment, as shown in the bottom corridor of figure 4.

Content Locations

To insert content items we can now assume a set of paths, each of which is associated with a visibility region. On each path we must then allocate m content items so they are all encountered (seen) on every path, and n content items that will be unique to each path. This allocation process is accordingly separated into two sections, one to place shared items, and one for unique content.

Shared content - Shared content is placed within the visibility region(s) seen on all our input paths. Given the visibility regions, this common polygonal environment is easily computed as the intersection of all such regions. Note that while having common starting and ending positions implies some overlap must exist, this can in general result in multiple, disconnected polygons.

To find a suitable location within these shared polygons, we first sort the set of shared polygons by area. This allows us to prioritize placement within relatively large regions, and so (heuristically) better ensure the item will be seen on each given path. Our polygons are not necessarily convex, and so placing an item within a polygon is not trivial. We thus find a *representative point,* a specific location guaranteed to lie within the polygon, by triangulating the polygon and selecting the centroid of the largest triangle. Subsequent placements of our m items descend down our polygon priority (size) list, and may also make use of different triangles within the same polygon. Figure 5 shows the final results for 2 shared content items, represented by two yellow circles.

Unique content - Unique content needs to be placed such that each of n items is only encountered on one of our paths. For this we can compute the difference between each visibility region and all others, and apply the same process as for shared content to locate items within the resulting set of polygons. Figure 5 shows the results for one unique content item (red circle) per path.

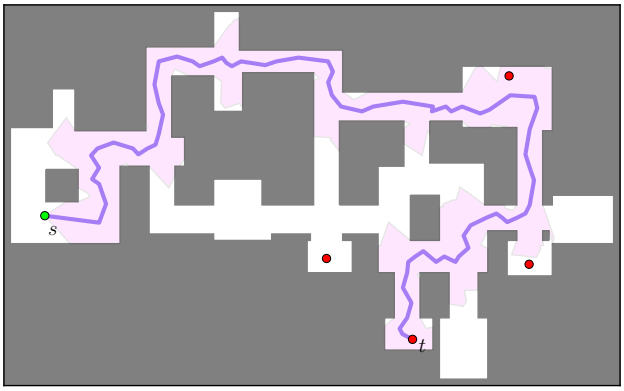


Figure 6: Wasteland level showing weak visibility region of a path

Content collision ratio - In more complex levels with many paths of interest, path overlap can easily be such that locations for content unique to each path may not exist. To deal with this, we also define a more flexible content collision ratio (CCR). This scalar value represents the maximum ratio of other paths on which a given unique content may be encountered. Given P total paths and a content which we intend to locate on one of those paths, a $CCR \in [0 \dots 1]$, allows a maximum of $\lceil CCR \times (P-1) \rceil$ other paths to also cover the content. A CCR of 1 then allows the content to be placed in a location that may be covered by all paths, a CCR of 0 will require the content be placed in a location that no other paths cover and hence is truly unique, and values between are proportionally permissive.

It is important to note that these content placement requirements are strict, and may thus fail for given geometries, path sets, and values of m, and n. The content locations will also of course be related to the FoV used, with different FoV assumptions, such as infinite range and/or 360° visibility, producing different results and making it more or less likely that satisfactory content placements exist.

Experimental Results

In this section we analyze the effectiveness of our approach on a non-trivial example level. We first verify that we can place shared, unique, and "never seen" content (the inverse of shared). Then we investigate the impact of these approaches through a small-scale human study. We implemented the method described in the previous section using python 2.7, the *Shapely* library which wraps the geometric engine - open source (GEOS) (The Toblerity Project 2015), and the *poly2tri* which provides Delaunay triangulation of polygons with holes (Hansen, Green, and Åhlén 2012).

Wasteland

In order to test our presented approach we modelled the *Temple of Titan* level from Wasteland 2 (inXile Entertainment 2014). Figure 6 shows this level with its multiple entry and

exit points indicated by red dots. Once entering this level the player has to reach the green position in order to advance the narrative and then exit the level through any of the red points. Note that we only modelled the general level structure, and did not represent game or puzzle elements such as doors, enemies, and collectables.

In order to show the potential of our method we explored placing content for a single entry and exit point (s and t respectively in figure 7, although technically s is not an entry/exit). Use of multiple starting and ending points increases analysis effort and can reduce opportunities for sharing content between paths, but does not change the basic approach. We consider placement of shared content, unique content using different ratios, and never seen content as a means of encouraging exploration.

Shared content - In designing a level, a developer may want all players to encounter some key content in the game. This only happens at the intersection of visibility regions of all the simple paths, which for a single start and goal minimally includes the start and goal nodes. Figure 7 shows 2 locations in yellow which all the player paths between s and t necessarily see. Note that the exact starting location was not selected as the resulting polygon was too small for the example content we considered.

Unique content ratio - When a level offers multiple overlapping simple paths, as shown in figure 6 it can be impossible to use our simple approach for finding truly unique content locations. In order to assure some uniqueness, but with more latitude and thus more likely to be successful, we use the CCR ratio previously presented. For this we used a brute-force approach where we check all possible combinations. For example, if the ratio is 0.5, a unique content is allowed to collide with half of the path visibility regions. We thus look at all possible combinations that use half of the paths or less until we have a candidate with an area region large enough to place the content item. In order to avoid having always the same region polygon returned for unique content, these combinations are shuffled for every request. This is in general an expensive process overall, but could be made more efficiency with a less brute-force approach, and capped to fail in place of excessive computation time. An example of the result of this process can be seen in figure 8, where unique content is placed for each path region option using a ratio of 0.5.

Never seen - In order to encourage exploration of the level, we can also look at the difference between the level polygon and the union of all the path visibility regions. The resulting polygons represent regions a player does not necessarily need to explore in order to solve the level. Figure 9 shows the results for the Wasteland level with never seen content (cyan points) placed in various dead-ends and near unused exit points.

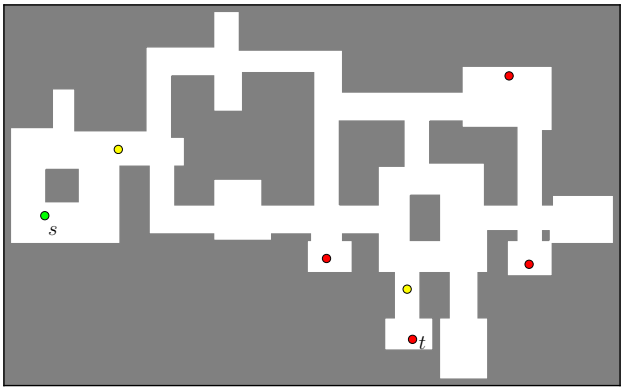


Figure 7: Two shared content (yellow dots).

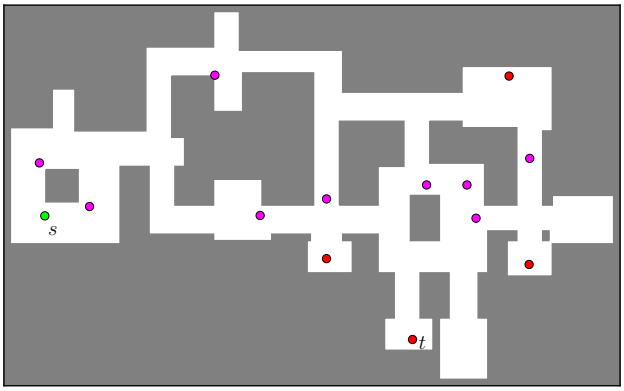


Figure 8: One shared content (magenta dots) per visibility path region, using a ratio of 0.5.

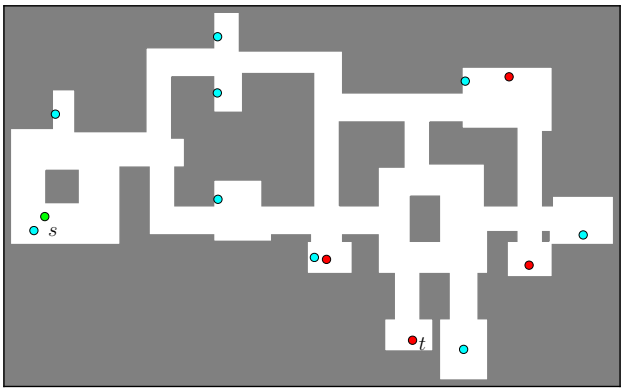
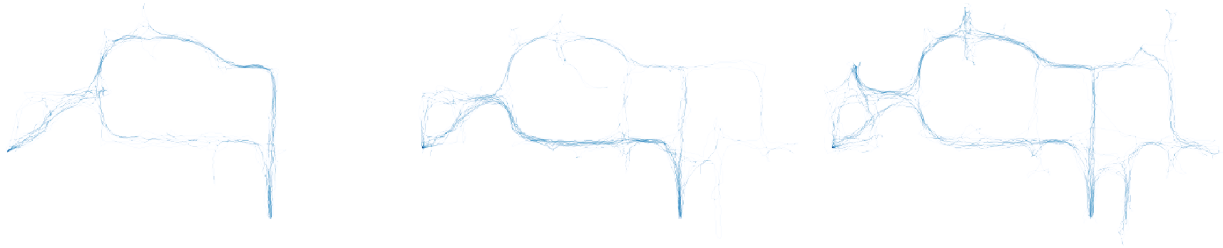


Figure 9: The never seen location (cyan dots).

Human Study

In order to evaluate the different presented methods, we ran an online study. We created a simple exploration game as a first person, 3D simulation, with the player required to find the exit from a fixed starting point. Each player was randomly assigned one of the three presented layouts representing different content distributions (figure 7, 8 or 9). Each content location had a unique art asset, *e.g.* an angel statue, barrel, well, *etc*. We had 73 participants drafted from the internet (*Unity3D* sub-reddit), where 20 played the shared content level, 26 the unique content with ratio 0.5, and 27 the never seen content level.

Figures 10a, 10b and 10c show the movement heatmaps for each layout. These results show interesting differences, although a larger data set is needed to establish statistical



(a) Shared content (b) Unique content (c) Never seen

Figure 10: Movement heatmaps for human experiment; for level geometry refer to figures 7, 8, and 9.

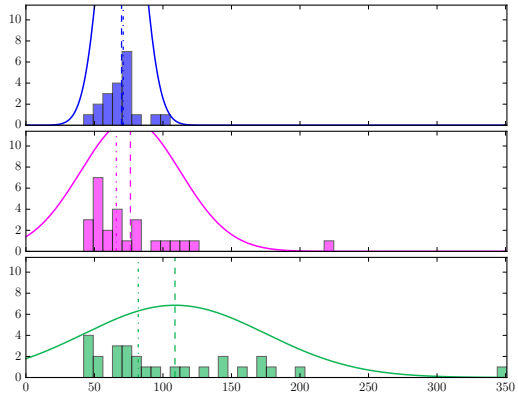


Figure 11: Distribution of time spent (s) in the level for each layout; the shared content (blue), unique content (magenta) and never seen (green) with their averages (dashed lines) and medians (dash-dot lines). The continuous lines show an equivalent normal distribution for the calculated mean and standard deviation.

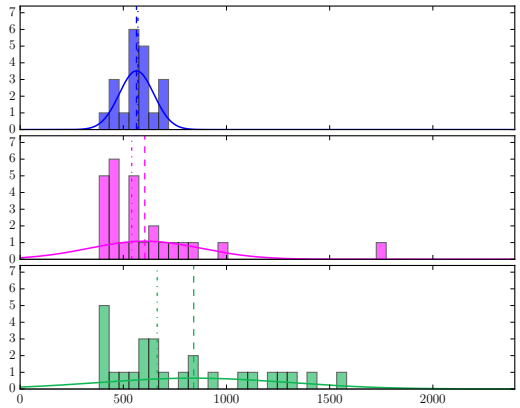


Figure 12: Distribution of the distance travelled (m) in the level for each layout; the shared content (blue), unique content (magenta) and never seen (green).

significance. In the never seen layout (figure 10c), we can see that players explored much more of the space than in the other layouts. This suggests that our design for placing content outside the main path options encourages exploration.

For a more quantitative view, we also looked at two other measurements: distance travelled and time spent in the level. Figures 11 and 12 show the distribution for each measurement. In general players with the never seen layout spent more time and travelled a longer distance than the others. Interestingly, the unique content case also seems to influence the players to move slightly faster towards the goal, as seen by a more skewed distribution and a median of 66s compare to 71s for the shared content layout. A similar observation can be made for the distance travelled. It is possible that unique content items acted as signposts, leading players along an expected path toward the goal. A larger scale study would help verify this, and additional experiments would also be useful to rule out the influence of other factors, such as the number of content items, which may also be influencing pathing decisions. Overall, though, we can see that all three layout achieve different outcomes in terms of players movement behaviour, and so our process for placing content has a controllable and potentially meaningful impact on the player experience.

Conclusions and Future Work

Observing game content, even narratively unimportant elements, is a major player motivation in most game genres. Leveraging that desire is thus an important aspect of game design, and processes to locate decorative content relative to likelihood of being seen are useful tools in encouraging exploration. Our visibility based design provides an elegant algorithmic approach to placing such content, with the human-study validation demonstrating that it has a measurable impact on player movement.

We are interested in further investigating the human motivations for path finding, as this information is essential in developing better positioning algorithms as well for movement simulations. Informed by human models, architects have multiple school of thoughts about where you should put circulation paths as well as content locations (Frederick 2007; Alexander, Ishikawa, and Silverstein 1977), and this general approach could be further exploited to make an improved, methodically defined game experience.

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

Alexander, C.; Ishikawa, S.; and Silverstein, M. 1977. *A pattern language: towns, buildings, construction*, volume 2. Oxford University Press.

Aonuma, H.; Imai, H.; Imai, K.; and Tokuyama, T. 1990. Maximin location of convex objects in a polygon and related dynamic Voronoi diagrams. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, 225–234.

Asano, T. 1985. Efficient algorithms for finding the visibility polygons for a polygonal region with holes. *Transactions of IECE of Japan* E-68:557–559.

Campbell, J.; Tremblay, J.; and Verbrugge, C. 2015. Clustering player paths. In *FDG'15: Proceedings of the 10th International Conference on Foundations of Digital Games*.

Dormans, J., and Bakkes, S. 2011. Generating missions and spaces for adaptable play experiences. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3):216–228.

Erickson, L., and LaValle, S. 2011. An art gallery approach to ensuring that landmarks are distinguishable. In *Proceedings of Robotics: Science and Systems*.

Frederick, M. 2007. *101 Things I learned in Architecture School*. MIT Press.

Ghosh, S. 2007. *Visibility Algorithms in the Plane*. Cambridge University Press.

Hansen, T.; Green, M.; and Åhlén, T. 2012. A 2D constrained Delaunay triangulation library. https://github.com/hansent/python-poly2tri.

Horswill, I. D., and Foged, L. 2012. Fast procedural level population with playability constraints. In *AIIDE'12: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

inXile Entertainment. 2014. Wasteland 2. http://wasteland.inxile-entertainment.com/.

Milam, D., and El Nasr, M. S. 2010. Design patterns to guide player movement in 3D games. In *Sandbox'10: Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games*, 37–42.

Smith, A. M.; Andersen, E.; Mateas, M.; and Popovi, Z. 2012. A case study of expressively constrainable level design automation tools for a puzzle game. In *FDG'12: In Proceedings of the 7th International Conference on Foundations of Digital Games*.

Suri, S., and O'Rourke, J. 1986. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the Second Annual Symposium on Computational Geometry*, 14–23. New York, NY, USA: ACM.

The Toblerity Project. 2015. Shapely — Python wrapper for GEOS, algebraic manipulation of geometry. http://toblerity.org/shapely/.

Tremblay, J.; Torres, P. A.; and Verbrugge, C. 2014. An algorithmic approach to analyzing combat and stealth games. In *CIG'15: Computational Intelligence and Games*, 301–308.

Winters, G. J., and Zhu, J. 2014. Guiding players through structural composition patterns in 3D adventure games. In *FDG'14: In Proceedings of the 9th International Conference on Foundations of Digital Games*.