Clustering Player Paths

Jonathan Campbell
School of Computer Science
McGill University, Montréal
Québec, Canada
jcampb35@cs.mcgill.ca

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

Player traces can result in a vast amount of high-dimensional data. This information is crucial to game designers trying to understand player behaviour and for tailoring the game experience. In this work we investigate different approaches to clustering player traces in order to expose useful game information. We consider three trace similarity metrics, as well as two clustering algorithms. We evaluate and compare the different approaches using trivial and non-trivial game levels from different game genres, giving guidance on clustering and metric choices, and illustrating the potential value of cluster information.

Keywords

Artificial Intelligence, Video Games, Stealth games, Clustering

1. INTRODUCTION

The development of digital games commonly requires an iterative process, looping between designing and play-testing. Information gathered from the latter is used to influence the design and produce a well tailored player experience. Play-testing information often takes the form of multi-dimensional traces, *e.g.* a player path from start to finish, where a state on that path has position, but also information about health, ammo, enemies alive, *etc.* Understanding such data is a complex activity, as the human behaviours of play-testers will result in a large amount of variance in the trace structure, even for relatively similar approaches. Techniques which aggreggate traces into similar groupings thus have significant value in helping designers more easily understand the range of game-play a level affords.

In this paper we present an analysis of the effect of automatically clustering player traces. Our approach is to use the full, high-dimensional space of player traces, considering not just the geometric positions but also other dimensions of measuring the player state, such as time, health, and so forth. In this way we provide a comprehensive and flexible view that can expose interesting differences in player traces beyond simple pathing choices. Our work is part of a bigger project, aiming to offer design-oriented tools which leverage Artificial Intelligence techniques (AI) to better understand the game experience.

We analyze two major clustering algorithms (K-Means++ and DBSCAN), along with three different metrics for comparing traces together (area, Fréchet and Hausdorff). We first seek to verify whether the selected clustering algorithms and metrics can produce reliable cluster results on test levels, and then through comparison determine which of each performs the best in terms of quality and computation time. We then apply our approach to multiple game-genres (combat, stealth and platformer) using non-trivial levels, to show some types of information that can be gleaned from clustering results. Our results help define useful choices in designing an efficient and effective clustering approach, and also illustrate the utility of considering multi-dimensional traces in understanding game-play. Specific contributions of our work include:

- We describe and explore different path similarity metrics and algorithms, taking a curve-oriented approach towards clustering player traces.
- The framework we use is explored in the context of Unity3D, and available as open-source, providing an industry-relevant and useful tool for further research [5].

2. BACKGROUND & RELATED WORK

Our approach in this work can be seen as part of the effort to offer design tools that leverage AI techniques to help design and create interactive content. AI-based design tools have a broad scope, encompassing problems such as path finding [17], procedural content generation [15], *etc.* Here we are interested in presenting data produced in testing (by human players or artificially) to game designers/developers to help them understand their game/level-design better. For this, simple statistics such as average time taken to complete a level, or average health loss, are common; we are concentrating on managing the complexity of more detailed analysis, grouping similar traces together whatever the dimension or metric of interest.

In order to produce the traces (paths) to be clustered, we used a random path generator for stealth [17] and platformer [16] games. The paths are produced using a randomized, heuristic tree-search algorithm (rapidly randomly exploring tree search) applied to a formal state representation, and respecting game rules. These traces could have

been produced by human players as well, but we are also interested in building game design tools that do not involve human players, using artifical traces as part of a solution to efficiently computing automatic game metrics [12] at design time. Our traces result in multi-dimensional data, recording position, time, and aspects of player state. Additionally, we also incorporate a measure of *risk* as an analytic metric previously presented by Tremblay *et al.* [18]. This latter measure is a stealth oriented metric, computing a heuristic risk factor for each point on a player path as a function of distance to enemies. This has been shown to correlate with human perception of risk, and thus represents an additional interesting and relevant factor in understanding path data.

Related Work - Clustering of player behaviours has seen increased interest in academic game work. Andersen *et al.* presented a genre-independent method called Playtracer to visually analyze play traces through a generalized heatmap, treating traces as sequences of game state [1]. Their approach required the designer specification of a state distance metric for each particular type of game, but in later work by Liu *et al.* these metrics were made easier to define by using state features to collapse states together without losing salient information [10]. Their focus is aimed at determining state pivots: moments in the trace where the players found a solution to the problem or where/when they gave up. Osborn and Mateas similarly created a trace visualization system, but compared sequences of input actions instead of state, using a special edit distance metric, in their Gamalyzer tool [13].

Our work revisits the state-based approach but looks at the state space geometrically, treating traces as curves – each point having a certain set of features (position, time, health, *etc.*) – and using curve-oriented distance metrics to compare them together.

In other related work, Bauckhage *et al.* presented a spatio-temporal clustering method for game levels [3]. Their method evaluates one player's in-game movement behaviours to determine relations between cover and non-cover choices and likely transitions between them. This work differs from ours as they are interested in the analysis of one path, whereas we are interested in grouping similar paths together.

3. CLUSTERING APPROACH

The clustering framework is composed of two major components: the clustering algorithm and the metric to compare traces together, which can be chosen independently. The input to the framework includes the set of traces, the dimensions to cluster on, and various parameters depending on the clustering algorithm chosen. The set of traces consists of paths, *i.e.* a set of points, which can have multiple pieces of information for each point, such as $\langle x, y \rangle$, time, distance to enemy, health, and other genre-dependent measures. Information about the level (such as walls or obstacles) can also be provided if cleaning of the traces is needed (see section 3.3 below). Mathematically, we are in interested in finding $p_i \sim p_j \ \forall p \in c_k$, where p_i and p_j are paths and c_k is a cluster. Note that a (theoretically) simple, intuitive interpretation of $\sim$ as homotopic equivalence is not sufficient, as while the level topology is a good guideline, not all obstacles induce meaningful cluster distinctions.

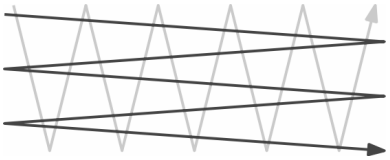

Figure 1: A small Hausdorff distance but a large Fréchet distance [19].

3.1 Metrics

Clustering algorithms necessitate a distance measure or metric between two objects in a dataset. In the basic case, when dealing with purely geometric coordinates of n dimensions, the Euclidean distance is used to compare any two points. Here, we are clustering on level traces, that is, sets of points (for our purposes), which can be generalized to geometric curves. Therefore, we consider curve distance metrics and their application to our context; below we discuss Hausdorff, Fréchet, and area (triangulation) similarity metrics.

The **Hausdorff** distance is a basic measure of similarity between sets. It is a maxi-min function H, which given two sets A and B determines, for each point in A, the smallest distance to any point in B, and returns the largest of those distances. The maximum of $H(A, B)$ and $H(B, A)$ is then taken to create a symmetric distance function [9]. Hausdorff distance considers all pairs of points, but is in practice relatively fast to compute, due to the low constant involved in performing the computation. Compared with our other metrics below, however, (and depending on the level context,) it also generally produces less desirable results, likely because it takes a set-based view of similarity, and does not take into account the ordering or location of points on the curve. This will be shown in our results discussed in section 4.

Fréchet distance is a metric that calculates the distance between two curves, directly factoring in the location and ordering of points on each curve. The common example is to imagine a man on one curve, walking his dog on the other. Both the man and dog follow their respective curves from start to endpoint and can vary their speed, but cannot go backward. The Fréchet distance is then the minimum length of the leash necessary to connect the man and dog at any point during their walk [8].

This metric has several advantages compared with some of the other metrics we considered. The most significant benefit of the metric is that it takes into account the ordering of the points on the curve, and thus acts as a better measure of curve similarity. Figure 1 shows the importance of this factor, illustrating two curves which are qualitatively quite different and have a large Fréchet distance, but which would be considered "close" in Hausdorff distance. For player traces where the distinction between a player who proceeds directly to the goal and one who does some amount of backtracking this may be quite important, and a metric like Fréchet is potentially a better choice. Fréchet also directly extends to curves in higher dimensions, and so like Hausdorff is useful in a multi-dimensional context, giving us flexibility in how we cluster. A disadvantage of Fréchet, however, is that a naive implementation is in practice slower to calculate than Hausdorff, a concern we partially solve by by using the optimized implementation provided by Buchin *et al.* [4]. Finally, note that the Fréchet distance comes in both discrete (weak) and continuous forms, and we are using the latter.

The **Area** between curves is another possible metric. Here, we calculate similarity in terms of the total area of the closed space between the two curves. With a monotonically increasing dimension (such as time), the shape between two curves is well defined, making area a good measure of similarity. In more abstract dimensions, however, or without a strong monotonic axis such as provided by the time dimension, the area between is not necessarily well defined (figure 1 is an extreme example in 2D), so while this metric has intuitive appeal, it tends to be limited to simpler 2D or 3D situations. As well, since our curves are defined by discrete points, we calculate the area between the curves by triangulating rather than integrating.

Note that throughout we make the simplifying assumption that paths progress from the same starting point to either a single end-point or a relatively small region of end-points. That latter in particular is not always the case, and in games where players may complete a level by satisfying more arbitrary constraints (such as jumping higher than a given threshold) a geometric interpretation of distance may not accurately represent logical similarity, and would require a more abstract distance measure.

3.2 Clustering

The second component of our framework is the clustering algorithm itself. Criteria for selection included speed, scalability, and quality of results. Algorithms also had to use a metric that compared distance between points, so that the trace distance metrics outlined above could be easily inserted. Both K-Means and DBSCAN were considered.

K-Means operates using the concept of cluster centroids: initially, centroids are picked at random from the datapoint space, and each datapoint is assigned to the closest centroid. The centroids are then recomputed based on an average of their assigned points, and the datapoints are reassigned again to the closest centroid, repeating until stabilization [11]. K-Means however does not always produce the optimal result; depending on the initial random choice of centroids, the algorithm can get stuck in a local minimum, which also decreases the performances of the algorithm. To fix these issues, Arthur & Vassilvitskii introduced K-Means++, an algorithm to seed K-Means with smarter cluster centroids by choosing each successive cluster centroid to be far away from the others with some probability. This addition was shown to considerably increase both accuracy and speed [2]. We used K-Means++ in our tests.

DBSCAN was also used as a clustering algorithm, with more success. Like K-Means, it deals with distances between datapoints. The algorithm has two inputs: minimum cluster size and a value ϵ. The idea is to look at the ϵ-neighborhood of each point, and if its size is greater than the specified minimum cluster size it forms a cluster. The ϵ-neighborhood of each point in that cluster is then checked, adding further points to the cluster if it is greater than the minimum cluster size, and in this way growing individual clusters. Points that have smaller ϵ-neighborhoods and which are not assigned to any cluster are marked as noise. DBSCAN works well in finding clusters of differing size, an area where K-Means is less successful. Furthermore, it finds the optimal number of clusters naturally as a function of ϵ, rather than having to specify a fixed number of desired clusters ahead of time. It also performs in most cases at least as well as K-Means in terms of speed, and is more configurable due to its two inputs [7]. In cases where a maximum number of clusters is desired, we also allow for K-Means++ to combine clusters returned by DBSCAN based on the proximity of their cluster centroids.

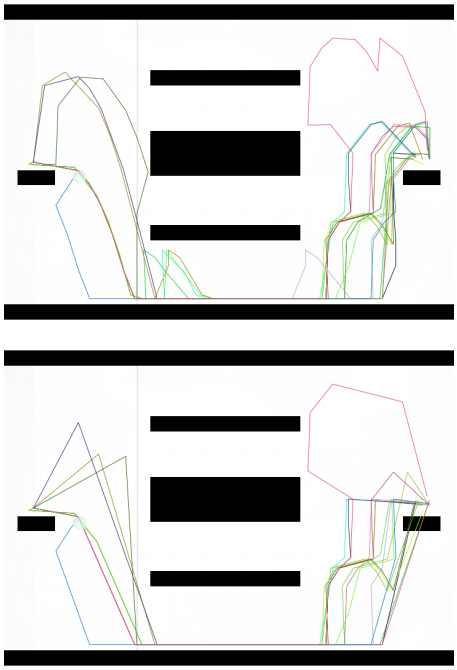

Figure 2: Before (top) and after (below) cleaning with RDP. Note how the general trajectory of each trace is maintained while unnecessary points are removed (dependent on a provided ϵ), while no path is smoothed to the point that it intersects the middle obstacles.

3.3 Trace Cleaning

Clustering can be an expensive process, and this can interfere with its practical usage in an interactive design setting. In our case the main cost is in computing path similarity, the cost of which is a function of the number of points used to define the paths. Recording the player's position in space and time at every frame, for instance, can result in a large number of points, many of which are very similar if not equal to the points adjoining them in the trace. In such cases, a path simplification algorithm such as the Ramer-Douglas-Peucker (RDP) algorithm [6, 14] can be used to remove extraneous points in the trace. RDP works by proposing a new curve composed of a straight segment from given start and end points, and either verifying that all points between in the original curve are not too far away, or including the most distant point as a necessary endpoint, splitting the proposed segment into two, and repeating recursively on the two smaller segments. Figure 2 shows before and after results, with the simplified paths following more or less the same trajectory, but consisting of far fewer points, and thus reduced clustering time.

In a game context, it is necessary for cleaned traces to continue to represent a feasible player path. For games that have obstacles – walls, or other known zones or states impossible for a player to access – a constraint was added to RDP to also not remove points in a path such that the resulting

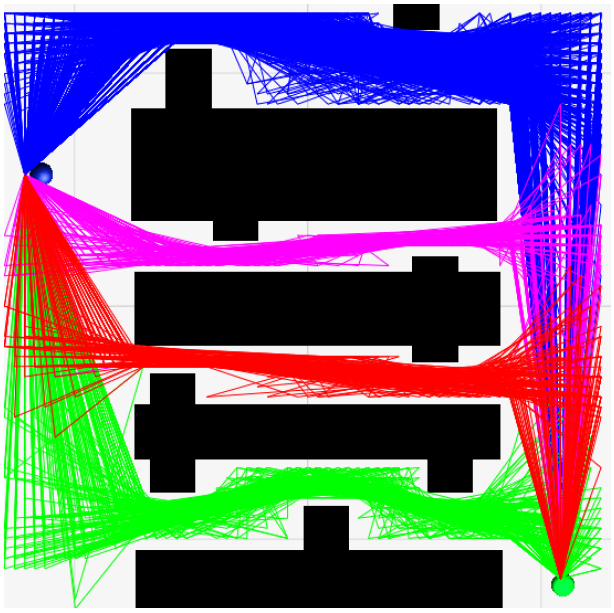

Figure 3: Overhead view of the Euclidean sanity level with 1500 traces coloured according to their clustering into 4 groups, using K-Means++ with the Fréchet distance on 2D coordinates. The blue (upper left) and green (lower right) circles represent the start and end positions of the traces, respectively.

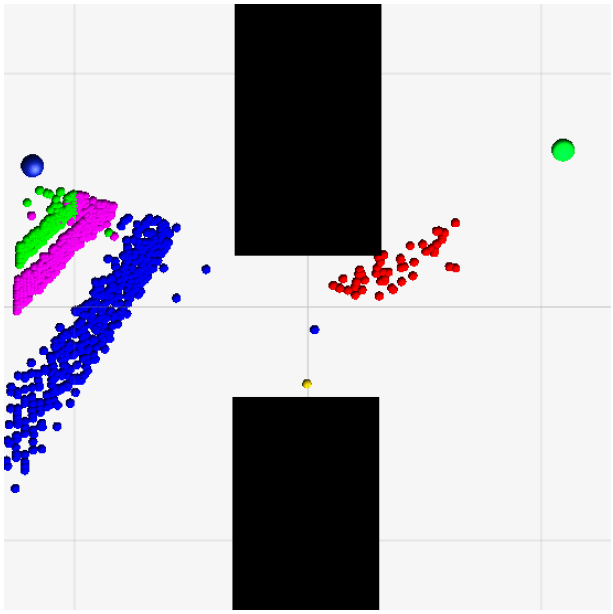

Figure 4: Overhead view of the time sanity level with 1500 traces coloured according to their clustering into 4 groups along the time dimension, using K-Means++ with the Fréchet distance.

path would pass through or go around said obstacle. This change was accomplished by taking the start and end-points of the line in the current RDP iteration, and then forming a triangle with each point that may be removed. If any vertex of any obstacle is contained within the triangle, the intermediate point must be retained. Note that in such case, the cleaning would require the positions of the obstacles in the level as an input.

4. EXPERIMENTAL RESULTS

In this section we explore the performance and possible applications of path clustering. We first present results on sanity check levels. These tests assure us that we are getting a satisfying output that matches our intuition of how paths should be clustered, independent of the numeric properties that drive the clustering algorithms. We then perform a comparison of the impact of the various similitary metrics and clustering algorithms, followed by an exposition of interesting results on non-trivial levels for different game genres. All traces used in this section were generated using the aforementioned random path generators [16, 17].

4.1 Sanity checks

A basic design goal for our clustering was for it to clearly identify trace groupings that represent significantly different game choices. Geometrically, this means respecting how paths navigate obstacles (homotopy), but it should also distinguish based on time, choice of game actions or behaviours. Below we show results using Euclidean traces, time-dependent traces, and gravity-influenced traces (platformer genre).[1]

Euclidean traces - Figure 3 shows a simple level with four distinct topological options from start to end position (excluding loops). As can be seen from the colouring, traces were assigned to distinct clusters based on the corridor they passed through. Note that we use K-Means++ here to show that it works in a simple context, and is sufficient in this context.

Time traces - For many genres, the time component is extremely important. In stealth games, for example, time dictates whether the player is seen or not by an enemy at a given position. In figure 4, an enemy is positioned at the small yellow circle in the central bottleneck and repeatedly rotates looking north then east, for a total of four times. This behavior creates four openings for the players to pass unseen by the enemy from the start on the left to the goal on the right, which are clearly identified in the clustering as four separate waves of player movement.

Platformer - In this example we used traces from the platformer domain, introducing the additional complexity of player motion being affected by gravity as well as manual player movement for left, right, and jumping. The platformer game used is one developed in Unity3D by Tremblay et. al [16]. Paths were cleaned using the RDP algorithm before clustering to reduce clustering time and get better results. Figure 5 shows our platformer test level with four clusters, mainly separated in terms of topological structuring, but also with some difference between the intial choice of jumping up or just falling (blue versus green clusters).

4.2 Variations

In this subsection we investigate the effect of our different path similarity metrics, followed by the choice of K-Means++ or DBSCAN.

Metrics - In the previous section, we presented three different approaches for measuring the distance between two curves: Hausdorff, Fréchet, and area. As these measures have different properties and costs, we are interested in knowing whether the metric choice has an impact on the clus-

[1]Videos are provided showing some time-dependent clusters and others that are hard to visualize using figures.

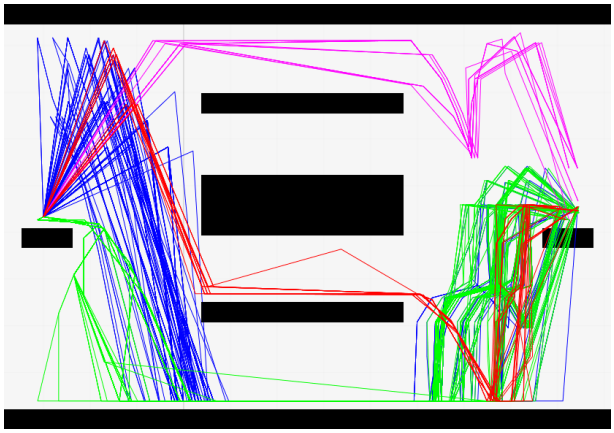

Figure 5: Side view of the platformer test level with 500 traces clustered on Euclidean coordinates into 4 groups, using DBSCAN with the Fréchet distance, after using the RDP algorithm.

tering quality. For this we hold the clustering algorithm constant, and compare behaviour of K-Means++ under different path similarity metrics.

For example, figure 6 shows a collision heat map of a K-Means++ clustering using Hausdorff distance on the Euclidean sanity level. Intensity of the blue coloring at each pixel is a function of the number of different clusters which paths passed through: darker means greater overlap, and thus less well-distinguished grouping.

In order to compare the different metrics, we ran 31 independent clusterings with K-Means++ with 4 clusters for each metric on the Euclidean and time sanity levels. For comparison purposes, we used fine-grain bitmap collision detection, and excluded the (necessarily) densely overlapping start and end sites by focusing only on the subset of the bitmap represented by the black rectangle overlaying the figure. Figure 7 shows the distribution of pixel collisions for all three metrics (area, Fréchet and Hausdorff), with their averages and medians. The presented order is from most to least stable metric. The figure shows that the area calculation is most stable, while Fréchet has about one to three wrong paths on average and Hausdorff is the least stable. Conversely, their computation time is inversely proportional to their quality.

We ran the same experiment on the time sanity level. For comparisons we used voxel collisions and a subspace of the 3D level (a cube) that again avoids starting and ending effects. The results are given in figure 8, showing the distribution of voxel collisions. The figure presents less convincing results than in the Euclidean case, as here while area is still best, Hausdorff has about the same average as Fréchet, and a much better median. All distributions, however, are much more spread out, with wide variance, and so a less clear distinction between clustering choices is not unexpected. Unlike geometric clustering, where topological structures help clustering make clear distinctions, a continuous time dimension does not impose homotopic properties, and even with our periodic guard rotations the cluster intervals have some degree of overlap.

Clustering algorithm - Our previous experiments used K-Means++ as one of the most popular and well known clustering algorithms. Other choices such as DBSCAN are

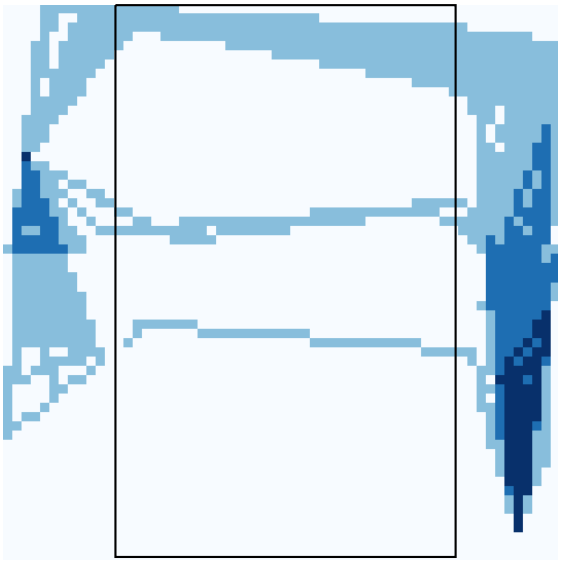

Figure 6: Cluster pixel collisions taken from the same level as shown in figure 3: the darker it is, the more paths from different clusters collide. The black-bordered rectangle is used to calculate the quality of the metric.

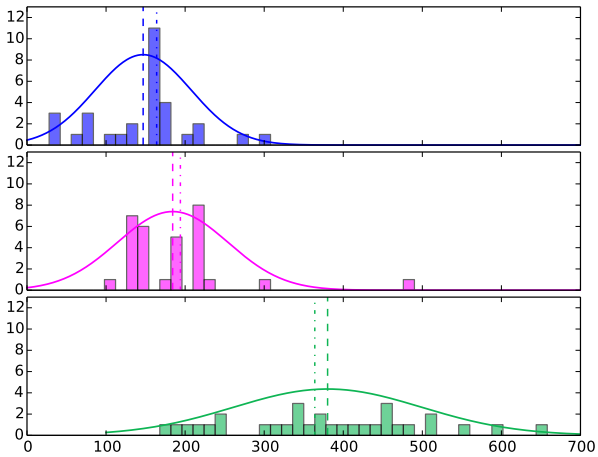

Figure 7: Pixel collision distribution of the 31 independent K-Means++ clusterings in the Euclidean sanity level for the three metrics: area (blue), Fréchet (magenta), and Hausdorff (green) with their average (dashed line) and median (dash-dot line). The continuous lines show an equivalent normal distribution for the calculated mean and standard deviation.

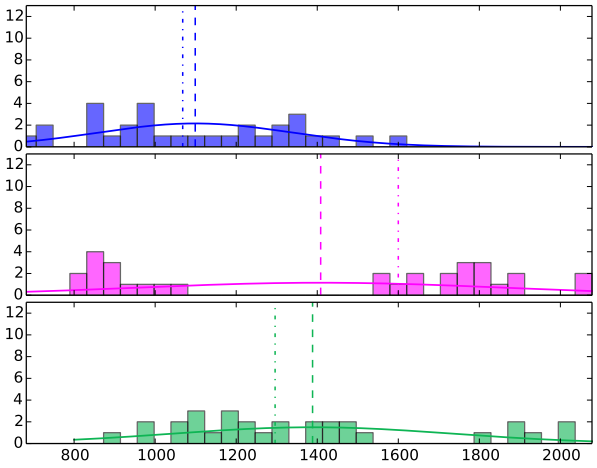

Figure 8: Voxel collision distribution of the 31 independent K-Means++ clusterings in the time sanity level for the three metrics: area (blue), Fréchet (magenta), and Hausdorff (green) with their average (dashed line), median (dash-dot line), and corresponding normal distributions.

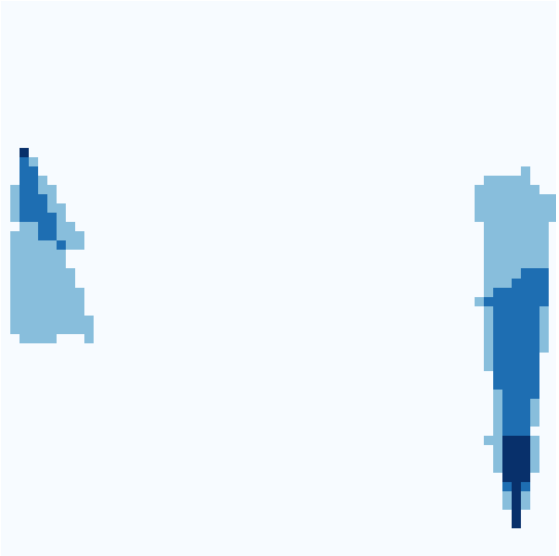

Figure 9: Pixel collisions of DBSCAN on the Euclidean sanity level.

of course possible. Algorithmically, and especially for our context, DBSCAN offers a number of advantages over K-Means++. The latter, for instance, requires the number of clusters to be specified as input to the clustering algorithm. This is not always convenient in player analysis, since it either requires ahead of time knowledge of the intended results, or an iterative reapplication of clustering to search for the best number of clusters. K-Means++ also assumes clusters should be of similar size, and while this is appropriate for evenly splitting up a dataset, it means traces indicative of interesting outliers or under-represented player strategies will be either diluted or spread into other clusters when enforcing the total number of clusters: understanding how most players will traverse a level design is important, but so is identifying unusual and less common groups of behaviours that may side-step designer goals.

Figure 9 shows the results of DBSCAN applied to the same traceset from our sanity level (figure 3), which was evaluated in terms of quality under K-Means++ in figure 6. In this case we use Fréchet distance as our similarity metric, as a reasonable compromise between quality and computational cost, and since we noted that the choice of similarity metric is much less critical under DBSCAN, with all metrics giving similar results. Here we see no cluster collisions are found in the middle of the level, although 156 out of the 1500 paths were labelled as noisy paths. Outliers thus do not distort the clustering, but must still be manually inspected if of interest.

A useful property of DBSCAN is that these results are generally deterministic—DBSCAN does not rely on random sampling, and so once a valid value of ϵ is chosen, the clustering always produces the expected results. This also indicates one of the main disadvantages of DBSCAN, in that the need to select a meaningful ϵ makes its usability by non-technicians non-trivial. It is also slightly slower than K-Means++. At the same time, we conclude that DBSCAN is the better choice over K-Means++ due to its more desirable results on our sanity level.

4.3 Non-trivial examples

In this subsection we will show non-trivial level cluster results in the stealth, combat, and platformer genres. Euclidean, time, and trace metrics will be used as cluster dimensions. These results demonstrate the various possible applications for our clustering framework, revealing possible information that may be useful to level designers.

Euclidean traces - Figure 10 shows a recreation of the first level of *Metal Gear Solid*, a popular stealth game and one that we used in several tests for our framework because its topology allows for distinct paths from starting position (top-left corner, blue sphere) to end position (bottom-right corner, green sphere), and which has important time and other metric properties as well. The figure presents the results of our default clustering approach (DBSCAN with the Fréchet distance), with each cluster taking a distinct set of corridors from start to end position with paths being homotopically equivalent, even though the clustering process had no knowledge of the level configuration.

Time traces - Figure 11 shows the results of clustering on the Metal Gear Solid level using geometric coordinates and time. Guards patrol throughout the interior of the level and so in the timestep at which the figure was taken there are no path nodes going through the central corridors. Once again, all clusters are clearly differentiated by space and time creating time-space dependent groups as seen by the characters (node) color. A level designer may be interested in these clusterings as indicating both path choice and the need to wait in order to bypass patrolling enemies.

Metric traces - It is also possible to cluster on abstract metric values associated with traces. Figure 12 shows the results of clustering on geometric coordinates and a *danger* metric (described in section 2) defined on each node in each trace. To better show the results we use an isometric view, projecting path nodes into the vertical-axis to indicate the amount of danger associated with a particular trace node. Notice that in this case clusters are located in semi-distinct

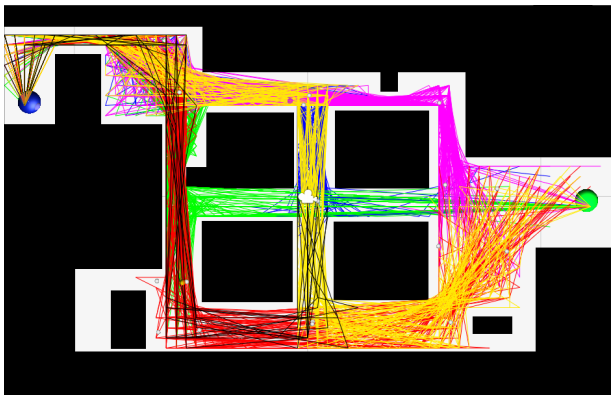

Figure 10: Metal Gear Solid's first level with 1500 traces clustered into 6 groups on Euclidean coordinates, using DBSCAN with the Fréchet distance. All clusters are visible in this static image except the blue cluster, which follows the top corridor, goes down through the central vertical corridor until reaching the middle, then follows the central horizontal corridor to the end position.

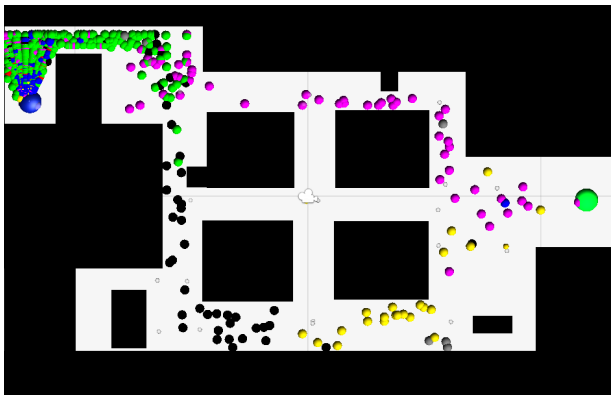

Figure 11: Metal Gear Solid's first level with 1500 traces clustered into 6 groups based on Euclidean and temporal coordinates. At the displayed timestep, we can see the purple cluster travelling through the top corridor, the yellow and black clusters going through the bottom corridor, and the green cluster still waiting to move out from the start position.

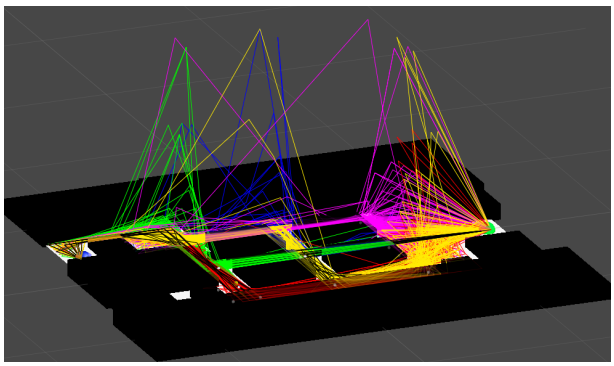

Figure 12: Metal Gear Solid's first level with 1500 traces clustered into 6 groups based on Euclidean coordinates and danger metrics of the nodes in each trace. We can see that the yellow cluster has high danger metrics towards the end position (right side), followed from right to left by the purple, blue, and green clusters. The red and black clusters, each taking a different set of corridors to the goal, are flatter and therefore have lower danger values.

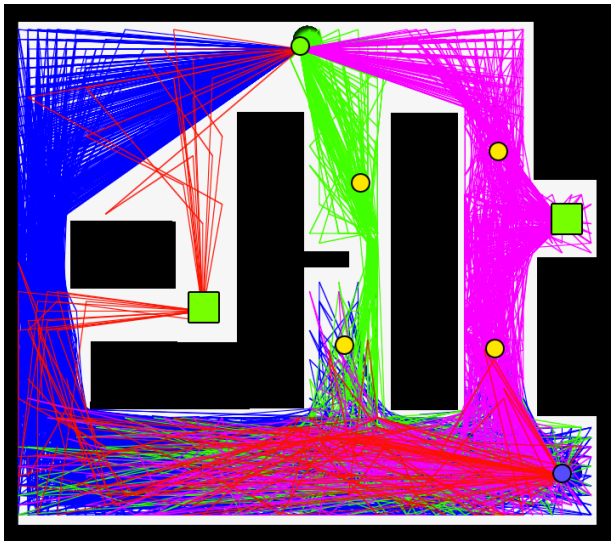

Figure 13: An experimental stealth level designed in Unity3D with two health packs (green squares) and four enemies (yellow circles), with 1500 traces clustered into 4 groups based on geometric coordinates and player health values, using DBSCAN with the Fréchet distance.

areas of the level, but also with differing degrees and distributions of danger values. The red and black clusters, for example, have almost no danger but follow different paths to the goal; taking either set of corridors is the least dangerous way to complete the level. Other clusters experience some amount of danger, but at different points—the green cluster has a high danger value early in the traversal, yellow primarily at the end, *etc.*

Another metric tested was the player health at any node in a trace. Figure 13 shows a level with two health packs (green boxes), and four enemies (yellow circles). The player starts at the blue circle (bottom right) and their goal is to reach the green circle (top center) alive. Using geometric coordinates and player health values we find 4 clusters which show traces that differ in path choice, enemies fought, and health packs used. The magenta group, for instance, went up the rightmost passage, usually fighting 2 enemies and picking up one health pack. The red cluster rather fought one or two of the bottom-most enemies and picked the health pack on the left before exiting. Green and blue clusters do not pick up any health packs, but fight 0–3 enemies, following different paths through either the central channel (green), or primarily along the bottom and left (blue, sometimes deviating to fight the lowest center-corridor enemy).

Player health is an important property. We thus also experimented with clustering purely on the health dimension. This resulted in 9 different groups, including different combinations of enemies fought and health packs taken. The choice of clustering dimensions is thus a flexible means for a designer to focus their understanding on pathing choices, time, player state, or any combination thereof.

Platformer - We also used our default approach on a non-trivial platformer level developed by Tremblay et al. [16]. In the level, seen in figure 14, there are two obvious paths from the start to ending position: either follow the bottom ledges and jump up at the end, or jump up at the start and follow the top ledges. However, it is also possible to follow the

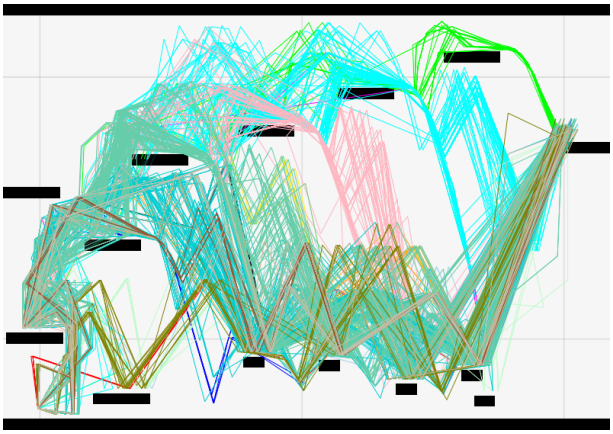

Figure 14: A platformer level with several paths from the starting to end position, showing 850 traces clustered into 17 groups based on Euclidean coordinates, using DBSCAN with the Fréchet distance. In-editor cluster selection/removal allow for interactive exploration of this static view.

top ledges for a certain amount, then drop down, follow the bottom ledges, and jump back up. Different choices of ϵ allow a designer to explore the coarse, overall behaviours, or the more subtle variations that depend on reaching different platforms or not. Figure 14 shows a detailed view consisting of 17 clusters.

5. CONCLUSIONS AND FUTURE WORK

A good understanding of player activity in a game level needs to be based on many traces, while also taking into account the multiple dimensions of game state that players tend to analyze. This kind of data easily results in an overwhelming volume to consider, even if much of it has little interesting variance. Appropriate clustering of traces in this high-dimensional context gives a useful means of finding and investigating the important, relative choices and trade-offs players make without drowning the designer in detail. Our approach – using DBSCAN in conjunction with the Fréchet distance – gives useful insight into this process, with the added advantage that our framework of investigation is open-source and built within Unity3D, and so of real and practical relevance.

An interesting direction for our analysis framework is in using it to find major trends in terms of player metrics, and applying that knowledge to train different controllers that mimic the major play-styles a level inspires. This would enable better automated analysis of future level designs. We are also interested in using the clusterings found in higher dimensional contexts to better understand the structure of more abstract measures of player activity—dimensions such as health, danger and so forth may not have hard-coded obstacles, but how clusters form with respect to these dimensions may still imply a heuristic, generated homotopy that reveals useful constraints on how a given level design can affect players.

Further work on developing more complex levels, as well as doing testing with developers in real-world game environments, would allow us to better establish the benefit of using trace clustering in level design. It would also be useful to compare the curve-oriented approach seen here with other, previously-seen trace representations, such as sequences of game states or actions, in order to determine the advantages and disadvantages of each approach.

6. ACKNOWLEDGEMENTS

This research was supported by the Fonds de recherche du Québec - Nature et technologies, and the Natural Sciences and Engineering Research Council of Canada.

7. REFERENCES

[1] E. Andersen, Y.-E. Liu, E. Apter, F. Boucher-Genesse, and Z. Popović. Gameplay analysis through state projection. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, pages 1–8, New York, NY, USA, 2010. ACM.

[2] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *SODA'07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[3] C. Bauckhage, R. Sifa, A. Drachen, C. Thurau, and F. Hadiji. Beyond heatmaps: Spatio-temporal clustering using behavior-based partitioning of game levels. In *CIG'14: IEEE Conference on Computational Intelligence and Games*, pages 1–8, Aug 2014.

[4] K. Buchin, M. Buchin, R. van Leusden, W. Meulemans, and W. Mulzer. Computing the Fréchet distance with a retractable leash. In *ESA'13: Proceedings of the 21st European Symposium on Algorithms*, pages 241–252, 2013.

[5] J. Campbell and J. Tremblay. Unity-3d — clustering open-source implementation. `https://github.com/GameResearchAtMcGill/unitytool/releases/tag/FDG2015`, 2015.

[6] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122, 1973.

[7] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[8] M. Frechét. Sur la distance de deux surfaces. *Annales de la Société Polonaise de Mathématique*, 1925.

[9] F. Hausdorff. *Grundzüge der Mengenlehre*. Leipzig Viet, 1914.

[10] Y.-E. Liu, E. Andersen, R. Snider, S. Cooper, and Z. Popović. Feature-based projections for effective playtrace analysis. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, FDG '11, pages 69–76, New York, NY, USA, 2011. ACM.

[11] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[12] M. Nelson. Game metrics without players: Strategies for understanding game artifacts. In *AIIDE'11: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2011.

[13] J. C. Osborn and M. Mateas. A game-independent play trace dissimilarity metric. In *FDG'14: Proceedings of the 9th International Conference on Foundations of Digital Games*, 2014.

[14] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244 – 256, 1972.

[15] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[16] J. Tremblay, A. Borodovski, and C. Verbrugge. I can jump! Exploring search algorithms for simulating platformer players. In *EXAG'14: First Workshop on Experimental AI In Games*, 2014.

[17] J. Tremblay, P. A. Torres, N. Rikovitch, and C. Verbrugge. An exploration tool for predicting stealthy behaviour. In *IDP'13: Proceedings of the 2013 AIIDE Workshop on Artificial Intelligence in the Game Design Process*, 2013.

[18] J. Tremblay, P. A. Torres, and C. Verbrugge. Measuring risk in stealth games. In *FDG'14: Proceedings of the 9th International Conference on Foundations of Digital Games*, 2014.

[19] R. van Leusden. A novel algorithm for computing the Fréchet distance. Master's thesis, Eindhoven University of Technology, 2013.