IADIS International Journal on Computer Science and Information Systems
Vol. 15, No. 2, pp. 76-90
ISSN: 1646-3692

ATLAS CHRONICLE: DEVELOPMENT AND VERIFICATION OF A SYSTEM FOR PROCEDURAL GENERATION OF STORY-DRIVEN GAMES

Elizabeth A. Matthews[1] and Juan E. Gilbert[2]
[1]*Washington and Lee University, USA*
[2]*University of Florida, USA*

ABSTRACT

Procedural Content Generation for Games is the application of computers to generate game content such as landscapes, vegetation, or maps. This paper proposes a system (Atlas Chronicle) for the generation of world maps for games that feature story-driven locations. The system uses a combination of story abstraction, graphs, physics simulation, Perlin noise, interpolation, and climate mapping to generate 2D tile-based maps given a basic story structure. To measure the fitness of the levels generated, a user study was performed using established game enjoyment metrics. The study utilized three different content generation approaches for comparison. Eighteen participants (11 Male, 7 Female), aged between 18 and 24 years old were screened based on their prior gaming experience. Results show that levels generated by Atlas Chronicle resulted in statistically similar responses as for the manually designed and static levels.

KEYWORDS

Procedural Generation, Video Games, Video Game Enjoyment

1. INTRODUCTION

1.1 Motivation

One of the efforts towards creating more enjoyable games involves the generation of content for said games. Traditionally, this content has been generated manually by human game designers and level designers. However, the two major problems with manual content generation for games are the expense (Takatsuki, 2007) and the fact that it does not scale (Iosup, 2011). Procedural Content Generation for Games (PCG-G) is the application of

computers to generate game content and select the enjoyable items for use in games (Hendrikx et al., 2013).

Procedural Content Generation (PCG), or Procedural Generation (PG) for short, is the generation of virtual content by computers, typically through a human-defined procedure. As games get larger and more expansive, a cheaper and faster approach to generating the game content is highly desirable. PG has been utilized in games for a long time, starting with one of the earliest games, Rogue (Epyx, 1980). A competition to generate procedural levels for the Mario game series (Nintendo, 2020) called the Infinite Mario Competition has been influential and one of the main motivators of researchers in this area (Shaker et al., 2011).

Games with PG can have a minimal amount of generated content, but the most prevalent and noticeable forms are repeated-play variations. Games like Diablo (Blizzard North, 1996) have the same storyline each time the game is started by a gamer, but the dungeons in which the player progresses through the story are procedurally generated. As a result, each time the game is played, the exploration is different. Although this effect is possible to achieve with manual content generation, it is unrealistic due to the sheer amount of time needed to create enough levels such that a gamer would not notice repeating levels after playing the game multiple times.

Traditionally, story-driven games have had minimal or compartmentalized procedural generation. Due to the innate relationship between the world map layout of locations of interest and the story to be told, the map is difficult to change without taking these story restrictions into account. The system described in this paper proposes a method that keeps the story structure intact but randomizes the directions one must travel to progress in the story. An abstracted story graph is used to implicitly build a map around the coordinates of the story locations.

1.2 Related Work

Procedural Generation has been utilized to create enjoyable games since the early 1980s (Epyx, 1980). The more traditional approach has been to generate a multitude of options independently of the user, while a more recent approach has been to dynamically generate the content based on the user's gameplay.

The procedurally generated content ranges in size and scope in the final game. Most commonly created are "game bits," which are the basic units that compose a game (e.g. textures or sounds), or "game spaces," which are the navigational world/space in which the player exists (Hendrikx et al., 2013). The systems which create the procedural content range from tools and materials for game designers to utilize in their normal workflow to independent designers and domain experts which fully design and evaluate the game (Khaled et al., 2013).

There are two fundamentally different times in the game development cycle in which PG techniques can be applied: (a) in the development of the game and (b) during gameplay. Independent Procedural Content Generation (IPCG) generates content without context from the player of the game, and Experience Driven Procedural Content Generation (EDPCG) generates content based on a user's performance and play style in the game. Independent Procedural Content Generation (IPCG) is the most popular version of PG, despite a recent rise in interest in research in EDPCG. IPCG generates many content variations, given a certain structure, then selects a variation for each play-through. The generation does not take input from the user. IPCG can be used to generate content from quests or puzzles (Pereira et al., 2016; Barros et al., 2016), platformer levels as in Spelunky or Mario (Baghdadi et al., 2015; Shaker et al., 2012; Yu and Hull, 2009), 3D terrain (Beckham and Pal, 2017; Frade et al., 2012), maps or mazes (Moore Jr,

2015; Togelius et al., 2013), creative sandbox games like Minecraft (Patrascu and Risi, 2016; Mojang, 2011), or infinite style games (Saltsman, 2009; Henschke et al., 2012). The technique we present in this paper in Atlas Chronicle is an IPCG technique, but there is no reason it could not be adapted for use in an EDPCG system.

2. METHODOLOGY

Story-driven games, often in the category of Role-Playing Games (RPGs), are one of the game categories that PG has barely touched due to the intertwined nature of the game world's map and the game story. We developed a system, Atlas Chronicle, that relies on the nature of RPG stories in which there is a partial order to the locations to visit. Further, the relative distances between story-based locations of interest remain within story-defined constraints of minimum and maximum distance and connectivity. The variance comes from the design decision to allow manipulation of (a) the cardinal directions between locations of interest, (b) the distances between locations of interest (within the specified constraints), and (c) their shapes and sizes.

If a system can calculate a controlled randomization of locations for each location visited in the game that satisfies the minimum and maximum distance between the locations as defined by the story, as well as the constraints of the game story's partial ordering, one can infer the world around these locations to build a full map. Our implementation of Atlas Chronicle finds this set of locations using a representation of the game graph and distance constraints alone. Atlas Chronicle uses a physics engine to place the locations of the story, a randomized floodfill to create the landmass around the locations, and a climate mapping to generate reasonable terrain changes across the landmass. Atlas Chronicle was developed in Python (2020) with the use of the PyGame module (PyGame, 2020) for visual and interactive aspects.

Maps generated in this manner can appear quite distinct, giving the player the impression of exploring a quite novel world each time the game is played, while nevertheless preserving all story-driven constraints specified in the game story's partial ordering. The impression of novelty could even be enhanced further by, for example, using miscellaneous psychological tricks at choice points, making one direction appear more obvious or more preferable than another in different iterations of the game, thus encouraging the player to take different paths through the story graph each time.

A game engine was designed to load the maps generated by the system for testing purposes. A user study was used to verify the quality of the generated results. The study utilized established video game enjoyment metrics to measure a decline of enjoyment over time in comparison to static gameplay (no map variation) and manually designed gameplay (human-created map variations).

2.1 Story Abstraction

Simplification is necessary to define a story-driven game in terms a computer can manipulate. In a story-driven RPG there are locations of interest (LOI) which the player visits. These LOI can be a town, cave, castle, or even a non-interactive terrain type, so the journey the player experiences travels through different terrains. Players may re-visit locations throughout the game, but there is an intended path through the world with travel distances between LOI designating the experience. A RPG story can be abstracted into LOI and distance restrictions

between each LOI. These restrictions are a story-experience restriction of distance: Town A and Cave B must be between 10 and 15 kilometers of each other. Restrictions are defined as a minimum and maximum value connecting two LOI.

These restrictions can be of two types: traversable or non-traversable. Traversable means that the restriction between LOIs A and B also implies as a rule that A must directly connect to B via traversable terrain. In our example, Town A and Cave B connected by a traversable restriction means that the direct path between A and B must be connected by walkable terrain. The "traversable" definition for terrain can change during the process of generating the map (see Section 2.4 for details). Sometimes LOI have restrictions between them, but not a direct traversable path (e.g., the final location must be at least 50 units away from the starting location). These non-traversable restrictions do not need to block the path between LOI but prevent LOI from being too far or too close to each other. Therefore, a non-traversable restriction does not need to be covered by non-traversable terrain; it simply does not require that the path be explicitly traversable terrain. This property is to allow restrictions to connect LOI to restrict their distances without affecting the connectivity of the terrain. Figure 1 shows a simple visualization of three LOI with three restrictions. In Figure 1 the intended progression of the LOI is from node A to node B, and then node B to node C. Node A and C are not directly traversable, but contain a restriction to keep them further apart.

The format of the story abstraction is stored in a JSON format, with nodes and restrictions defined with data which will be used in the later steps. Figure 2 shows a simplified sample of what the JSON format is. Dimensions are the min/max values for the area in which to place the LOI, the nodes are the LOI, and the restrictions use the keys from the nodes data to link two LOI. The min and max values for the restrictions indicate the story-based minimum and maximum distance between two LOI, and the typed boolean is the traversable flag.

With this abstraction of the story elements, the world can be built around the story. Combining LOI and restrictions puts the story in the format similar to a connected graph, one which the computer can use to design a map. The first step is to find some set of coordinates that satisfy all the restrictions in the connected graph of LOI. Unlike most graph planarization research, restrictions in the story abstraction can cross one another. This is due to the fact that paths often cover similar terrain from one LOI to another. The non-traversable restrictions are not to enforce the impossibility of traveling between two locations, merely to keep them separated by a certain amount. As long as the landmass generation preserves the traversable restrictions in later steps, crossing of restrictions is not considered a flaw (see Section 2.3). The property which matters for the placement of the nodes (LOI) is that the relative distances match the restrictions between LOI. The settling of LOI in these satisfactory coordinates uses a custom physics engine.

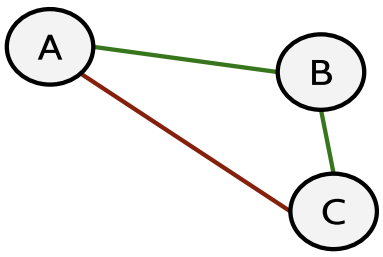


Figure 1. Example of story abstraction with three LOI represented as circles

```
{¶
···"dimensions":·{·¶
·····"height":·{"max":·55,·"min":·40},·¶
·····"width":·{"max":·50,·"min":·35}¶
···},·¶
···"nodes":·{¶
·····"DesertPalace":·{¶
·····"climate":·{"humidity":·0.5,·"temperature":·0.8}¶
·····},·¶
·····"DesertTown":·{¶
·····"climate":·{"humidity":·0.5,·"temperature":·0.8}¶
·····}¶
···}¶
···"restrictions":·{¶
·····"restrict1":·{¶
·····"dimensions":·{"max":·15,·"min":·10},·¶
·····"first":·"DesertPalace",·¶
·····"second":·"DesertTown",·¶
·····"typed":·true¶
·····}¶
···}¶
}¶
```

Figure 2. Example story abstraction file for two LOI and one restriction. The "typed" property of a restriction is if it is traversable

2.2 Physics Engine

A 2d rigid body physics engine (built on Pymunk, 2019) applies forces on LOI as physical objects to push and pull the LOI into coordinates that satisfy the restrictions. Each LOI was represented as a physical object, and each restriction was made into a slide/spring connector between physical objects. The slide spring connection functions as a free movement sliding joint when within valid distance measurements and a spring outside the distance restrictions. If the connection distance/size is less than the minimum value of a restriction the physical property is changed to a spring-style connection with a resting size of the minimum value. The forces applied by the spring will push the two connected LOI objects further away if the distance connecting them is too small. A similar physics interaction happens if the distance is larger than the maximum value with the force applied to bring the two LOI closer to each other. The slide spring changes to a spring whose resting state is that of the maximum distance defined by the restriction and attempts to pull the two LOI nodes closer together. A visualization of this process is in Figure 3.

With all LOI added to the physics system and connected by slide-spring restrictions (minimum and maximum distances based on the story parameters), each LOI was given a random x, y coordinate in the physical plane. The size of the physical plane was also user-defined, like the story abstraction, with a minimum and maximum area. Once in the physical plane the system simulated the forces until all the objects' velocity fell below a small threshold. This settling pushed and pulled until the distances between all LOI were close enough to their valid range such that the velocities grew close enough to zero to be considered "stopped."

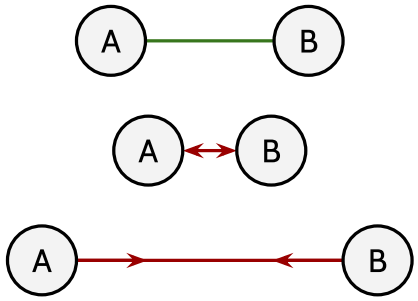


Figure 3. Top to bottom slide spring states: no force applied, increase distance force applied, reduce distance force applied. Green indicates no forces applied and red indicating forces applied away (middle) and towards (bottom)

The use of a physics engine over other more deterministic algorithms was to account for human error in the story abstraction step. Future implementations of the system will be available for game developers and a non-deterministic allowance will allow for imperfect systems to still create solutions. The final result is shown in Figure 4a.

2.3 Terrain Generation

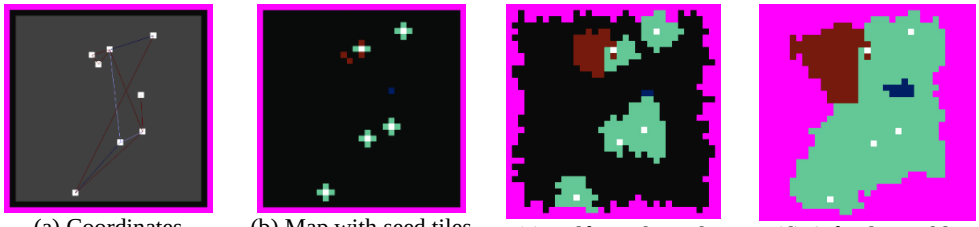
Once each of the LOI has valid x,y coordinates that satisfy all restrictions to the best of the system's ability, the terrain around the LOI is generated. A randomized floodfill fills in terrain around the LOI. Each LOI was given a seed terrain. The terrain-based LOI were a single seed terrain tile and each interactable/town-based LOI creating four seeds around it of terrain. Users can define the types of terrain around each town to the four main cardinal directions or randomly (e.g. a fishing town can have water next to it). The edges of the map were filled with void tiles, which are tiles that will be replaced in further iterations. The use of void tiles creates an organic look to the final resulting map. Figure 4a shows the result from the physics engine handed to the floodfill process and Figure 3b shows the seeded tiles added to the map.

At this stage the system begins the process of filling in the passive areas of the map (those tiles not determined by the seeded terrain values from the LOI). Each of the seed tiles and edge void tiles from Figure 4b were placed in a list, A, to be used in the floodfill sequence. The floodfill algorithm, Figure 4c, was as follows:

1. Select a tile t from the list of available tiles A
2. Populate list T of possible floodfill adjacent tiles from t
    a. Check all cardinal directions North, South, East, West of t
    b. Add tile to T if it is unfilled and viable
3. IF no tiles are in T, remove t from A and return to step 1
4. ELSE select one tile t^1 randomly from T
5. Fill t^1 with the same tile type as t and add t^1 to A
6. Repeat steps until A is empty

Viable means that attempting to fill the tile cannot cross a boundary along the line of any traversable restriction with a not-traversable tile type. Void tiles, mountain tiles, and water tiles are restricted in this version of Atlas Chronicle as not-traversable for this step. Any two LOI

connected by traversable restrictions will have a map that connects them via traversable terrain. The final result was a map of traversable terrain (“land”), not traversable terrain (“mountain” or “water”), and void tiles (to be filled later), Figure 4d.



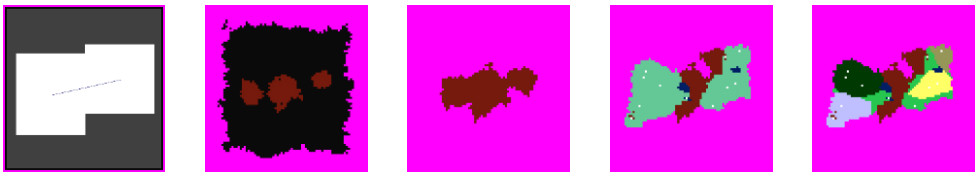
(a) Coordinates determined with physics engine

(b) Map with seed tiles based on LOI definitions

(c) Halfway through the flood-fill process

(d) A final possible map given LOI and restrictions

Figure 4. The process from the coordinates generated by the space manager to 2D map. Void tiles are pink, traversable terrain is green. Non-traversable appears in two formats: mountains (maroon) and water (blue)



(a) Physics engine with two fields

(b) Halfway through floodfill process

(c) Final mountain shape created

(d) Placement of fields

(e) Climate mapping (Figure 7) applied

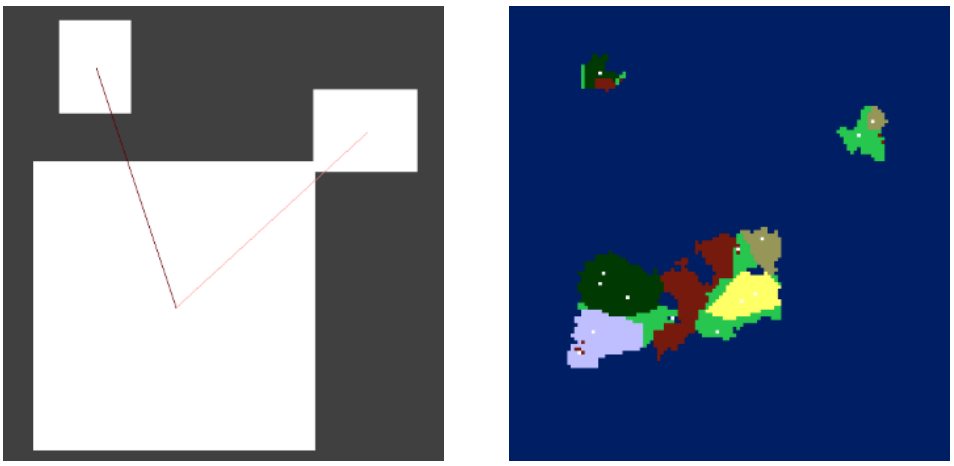
Figure 5. Recursive process for continent ending with one possible continent generated

2.4 Recursive Process

The next step repeats the process from Section 2.3 recursively. The traversable map generated, Figure 4d, is called a “field.” A field is a subset of the tiles that make a continent, which is a subset of the tiles that make a world. A field is a set of LOI separated by traversable or not traversable terrain. A continent consists of fields separated by not traversable terrain (mountains in this version of Atlas Chronicle). A world contains continents separated by water.

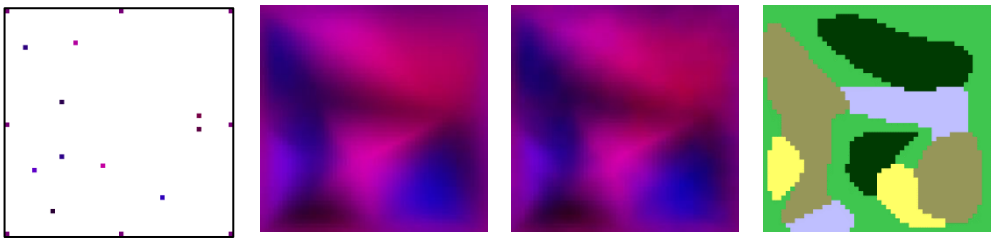
The next iteration for creating a continent repeats the process with fields in place of the LOI (Figure 5). Instead of a 1x1 block representing LOIs, each white rectangle was set to the dimensions of the field generated from the previous step. The fields were connected by restrictions, placed in a physics engine randomly and allowed to settle (Figure 5a). The floodfill has a similar process with void tiles around the edge, but the floodfill seed tiles were mountains instead of landfill (Figure 5b), to ensure the two or more fields were connected by mountains, thus creating a single continent (Figure 5c and 5d). The seed tiles are placed halfway between each field in a continent and in the center point of each field, instead of next to the LOI. Figure 5e shows the climate to terrain mapping, covered later in this section.

The final recursive step repeats the process with continents instead of LOI (Figure 6a). The dimensions of each white rectangle were based on the finished continent dimensions. Instead of floodfilling a third time, for this study's purposes the map is complete, and the unfilled areas were filled with ocean/water tiles (Figure 6b).



(a) Physics engine with three continents (b) Final possible world creation

Figure 6. Recursive process for world ending with one possible world generated



(a) Seed and edge values (b) Interpolation (c) Noise added (d) Terrain mapping

Figure 7. An example of mapping seeded climate values to a full map via interpolation. The final steps translate the two-dimensional climate to a terrain via the Terrain Boundary Map (Figure 8)

2.5 Terrain Mapping

After the continents were formed, the system mapped terrain types onto the landfill tiles (Matthews and Malloy, 2012). A climate was assigned to each LOI. Climates are made of a value between 0.0 and 1.0 for both temperature and humidity. A climate map was made by first creating seed values from each LOI and an average value was added to the edges (Figure 7a) to enable interpolation (SciPy, 2019) to the edges of the map. Then two interpolation maps were

generated to interpolate the climate values in the unassigned tiles between LOI, one for each temperature and humidity, visualized as blue and red respectively in Figure 7b. Noise (noise 1.2.2, 2019) was added for a more natural shape of terrains in Figure 7c. Then the climate was mapped to a corresponding terrain type based on a look-up process called a Terrain Boundary Map (TBM) in Figure 8 resulting in Figure 7d.

The system achieved the climate to terrain mapping by creating a nearest neighbor map to the designer's specifications, the TBM (Matthews and Malloy, 2012). For this study the map seen in Figure 8 was used, but any amount of terrain values can be used. The values for temperature and humidity were used as a x, y coordinate look-up in the nearest neighbor map, which converted the two values into a single terrain type.

The resultant map generated by Atlas Chronicle is saved in a JSON type file as seen in Figure XX. For a game engine to use the created content it needs to read in each tile type from the file as well as the list of interactable LOI for mapping game interactions. A simplistic engine was designed to verify and then run user studies with the Atlas Chronicle maps.

2.6 Testing the Game Engine

The Atlas Chronicle system was used in a series of user studies in an attempt to measure the difference in game enjoyment between games with procedural generation, and games without it (Matthews, 2019). The ability to collect survey information (described below) on the user's experience was also added.

The intended play-time of a normal RPG is 40+ hours, which was not feasible for a rapid, repeated test in a user study. Instead, for testing the procedural generation in RPGs used a minimal game engine and storyline created for the study. The engine was also programmed in Python/Pygame. A screenshot from the game is seen in Figure 10. The game engine contained the minimal elements to progress through the story: Overworld movement, locked terrain type travel, text boxes/interactable locations of interest, and gating items obtained from locations of interest.

The participant controlled the main character with a directional pad on a controller, initially limited to walking on traversable terrain only (not allowed to traverse over water or mountains). Standing on top of an interactable LOI and pressing a button opened a textbox describing what people in the town said to the main character along with hints as to where to go for the next story progression. The directions in the text boxes were dynamically created by the relative positions of LOI ("NORTHEAST", "SOUTH", etc based on the direction). The gating items were given to the player once they interacted with a location containing the item (e.g. The Desert Palace gives the player the Crystal of Courage). Certain locations would not permit further progression until the correct gating item was obtained. These gating items were used to simulate the traditional gating behavior of RPGs, where some challenge is required to "open up" the next exploration area. These are not handled by Atlas Chronicle but an intended human consideration when providing the system with the story abstraction. Eventually the game will give the player the "ship" item, which permits travel across the ocean.

Multiple generated maps were saved as JSON files and the game system would pick randomly from the available files. The map generated by Atlas Chronicle was converted to a two-dimensional game via Python and PyGame. The game engine kept track of the time elapsed in-game and would pause the game automatically at timed intervals to collect survey information based on established game enjoyment metrics. The game would only be allowed to continue after the participant completed the survey.

```
{¶
···"tiles":·[["water",·"water",·"mountain"],¶
·············["forest",·"grassland",·"mountain"],¶
·············["water",·"mountain",·"water"]],·¶
···"size":·[137,·135]¶
}¶
```

Figure 9. Example final JSON file created by Atlas Chronicle

2.7 User Study

Figure 10. Example gameplay of the 2D RPG testing engine for Atlas Chronicle

The terms “replay value” or “replayability” are often used when describing the benefit of procedural generation in games. We define the concept of replay value formally, with respect to the use of this term to describe video game enjoyment, as the rate of enjoyment decline over time spent playing the game. There are established methods for measuring enjoyment in video games using subjective style questionnaires. One frequently used questionnaire, the Game Experience Questionnaire (IJsselsteijn et al., 2008), has been thoroughly tested (Kätsyri et al., 2013; Mekler et al., 2014; Johnson et al., 2015) and is easily applied to measuring enjoyment in PG applications. The Fang et al. Questionnaire was also used because it is reliable way to measure Affect (Fang et al., 2010). Both the GEQ and Fang et al. Questionnaire were used for this study.

Participants were selected from applicants with some expressed interest in playing video games and were screened using a brief screening form. The screening form asked for the applicant’s self-reported gamer level and to select which types of games they regularly enjoy. Applicants who selected RPGs as a game type they regularly enjoyed were selected to participate. If applicants did not indicate they enjoyed RPGs but labeled themselves as an expert

or frequent gamer, they were invited to participate as well. A total of 18 participants (11 Male, 7 Female), aged between 18 and 24 years old (average age of 20), were selected for the study.

The user study included two phases: Phase 1 evaluated an infinite runner type game that was generated by a separate procedural generation system (Matthews, 2019). Phase 2, described in this paper, measured enjoyment of static, manual, and PG (Atlas Chronicle) versions of the RPG. The same participants were used for both phases to allow a within-subjects comparison. Due to the length of time necessary for each phase, the two phases were accomplished on separate days, with half of the participants performing phase 2 before phase 1.

Participants were assigned the three content generation formats in random order: static, manual, or procedural. A Latin square was used for the randomization. For the procedural and manual formats, each successive game that was played was different, with the creation of the content created by the computer or human designers, respectively. The number of pre-created levels for each type was sufficient to guarantee no two levels were repeated during the length of the study based on average time to complete a game and time dedicated to the phase. The static format game was replayed using a single created level for all games played.

Participants completed successive play-throughs of the RPG for half an hour per generation type. Due to the rapid iterative process, the subjective questionnaire enjoyment measurements were recorded exactly five times instead of after each game iteration. The RPG would play for 6 minutes before displaying the questionnaire, repeated for a total of five times per generation type. The time was decided after a pilot test determined the average play time a single game of the simplified RPG would take. The data collection was contained within the computer rather than on paper or at a different digital location to minimize player removal from the game-play experience. After the fifth questionnaire was completed the participant was informed to come find the observer and take their stretch break (a brief walk and restroom access). After the break, the participant was switched to the next content generation type.

2.8 Analysis

The independent variables were the three level generation methods: static, manual, and procedural. The dependent variables were the subjective questionnaire values: The 14 questions from the In-Game GEQ Module and the 5 questions in the Fang et al. Questionnaire. User responses were recorded periodically during each Game/Generation section. The responses were on a Likert scale from 0 to 4, which are visualized by color in Figure 11. To detect patterns, a contingency table was built upon varying factors and then the results were tested for statistical significance. To illustrate the results, this paper focuses on the GEQ question 3, “I felt bored,” and will not discuss results from all 14 + 5 questions on the survey. For full analysis of all questionnaire parts, see our previous work (Matthews, 2019).

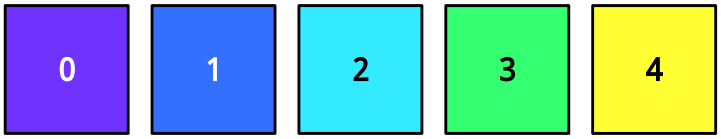


Figure 11. Color key used in the subjective contingency tables

The first test applied on a contingency table was the Chi-Squared test. GEQ03, “I felt bored,” was examined by segment number. The p-value was less than 0.01, therefore, the null hypothesis was rejected, and it was concluded that there was a relationship between segment number and response values for GEQ03. The questionnaire data are discrete rather than continuous; therefore, nonparametric tests were needed. Due to repeated significance testing, the chances of a Type I error ("false positive" or rejection of a true null hypothesis) are elevated, and hence significance values for each test were adjusted to reflect the true probability under the null hypothesis. The Holm-Bonferroni p-value adjustment method was applied to counteract this effect.

For visualization, colors are used to distinguish the different response values, with the color key labeled in Figure 11. Seen in Figure 12a, the numerical totals are represented as a proportion of the total responses per segment. A larger segment indicates a larger percentage of the total. The upper x-axis is used to indicate the segment number and the lower x-axis is used to indicate the pairwise results. The chi-squared p-value is in the upper right and the Wilcoxon results as labels are on the bottom axis. The results from the chi-squared test and nonparametric pairwise tests indicated that that boredom, as measured by GEQ03, increased significantly from the first segment (Segment 1) to the last segment (Segment 5) during the test.

Given that there appeared to be a general increase in boredom over time, the analysis compared the early and late responses, focusing on the survey responses from segment 1, collected after the participants had played the game for the earlier designated amount of time, and segment 5, which were collected after playing the generation type for the full amount of time (Figure 12b). Generally, the pairwise analysis indicates that segment 1 for the manual designs was “better” than segment 5 of the static and procedural designs regarding negative affect scores. But all other categories were too similar to differentiate. However, one can conclude that the segment 1 (start) values for all three generation types were similar, and all three segment 5 (end) values were similar. The conclusion is that procedural was no better and no worse than any other content generation approach.

3. CONCLUSION

Atlas Chronicle is designed for ease of adaption and human error when designing story-driven game maps. By using a story-abstraction to design a physics system, the system adjusts for human error to find a solution for LOI placement by distributing forces across the conflicting parts of the system. Atlas Chronicle uses a seeded climate interpolation to create smooth transitions between differing terrain types, inferring the terrain type from the climate with the Terrain Boundary Map. The recursive process enables the system to be extended to more layers of connectivity in the world design.

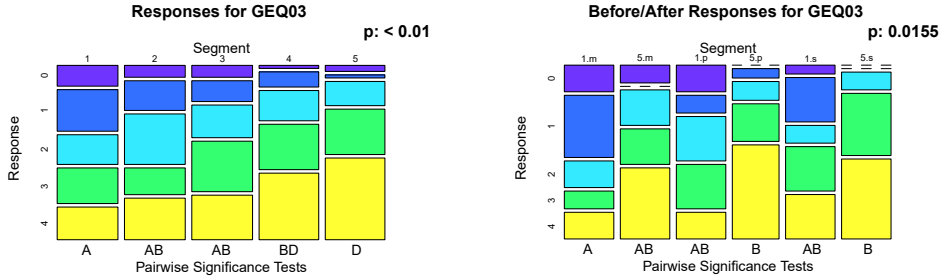
Based on the analysis, the boredom question from the GEQ measured a decline in enjoyment, over time. However, when comparing a before and after approach between the generation types, the procedurally generated test set performed the same as the manual and static design sets. This does call into question the claim of procedural generation increasing replay value, but for this research we can conclude that Atlas Chronicle is no better and no worse than static or manual generation approaches. According to our results, if a game designer wanted to have multiple, varied levels for their game, Atlas Chronicle would be a viable option because no enjoyment

would be lost by using procedural generation, and procedural generation is faster and more cost-effective than manual procedures.

Future enhancements to the system include mapping on a three dimensional plane, smoothing of edges between different terrain type tiles, and integration with more complicated game engines. Translation from a two dimensional map to a three dimensional map in games is not as daunting as it might seem. Most games with three dimensional graphics and navigation still operate on a fundamentally two dimensional space due to gravity keeping the player on the ground. The addition of a noise-generated height map, with control such that terrain like "mountains" are appropriately scaled up, is enough to apply Atlas Chronicle to most three dimensional story-driven games. Edge smoothing of tile-based games require a neighboring tile search to decide what percentage of the tile would be covered by land, water, etc.

More complicated game engines can be wholly unique, for testing, research, or indie game development, but not the only systems Atlas Chronicle can help. Established game franchises which focus on player exploration and discovery of locations of interest can use Atlas Chronicle with little modification. For example, the Pokémon series (Nintendo, 2020) typically has overworld style terrain with locations of interest gated by unwalkable terrain. The Pokémon series already has emphasis on exploration to find new types of monsters to befriend, which can be made into a fresh experience with unique procedurally generated maps from Atlas Chronicle.

We revealed statistically relevant conclusions about the procedurally generated maps created by the Atlas Chronicle system. In summary, the maps generated were no different than human-designed maps in terms of retaining enjoyment over time. Participants experienced the same decline in enjoyment over time regardless of the content generation used.



(a) Overall responses for boredom over all five segments

(b) Responses at the start and end of the test, pooled by generation type

Figure 12. Visual contingency tables using the color key from Figure 11 for the GEQ question "I felt bored." Y-axis is the response value from 0 (not at all) to 4 (extremely). X-axis on the top labels the segment for which the response was recorded, with m/p/s for manual/procedural/static. Bottom X-axis labels the pairwise analysis: columns which do not share a letter are significantly different

REFERENCES

Baghdadi, W., Eddin, F. S., Al-Omari, R., Alhalawani, Z., Shaker, M., & Shaker, N. (2015). A procedural method for automatic generation of spelunky levels. In European conference on the applications of evolutionary computation (pp. 305--317).

Barros, G. A., Liapis, A., & Togelius, J. (2016). Playing with data: Procedural generation of adventures from open data. In Digra/fdg.

Beckham, C., & Pal, C. (2017). A step towards procedural terrain generation with gans. arXiv preprint arXiv:1707.03383.

Blizzard North. (1996). Diablo (pc game).

Epyx. (1980). Rogue. Game [PC].

Fang, X., Chan, S., Brzezinski, J., & Nair, C. (2010). Development of an instrument to measure enjoyment of computer game play. Intl. Journal of Human--Computer Interaction, 26(9), 868--886.

Frade, M., de Vega, F. F., & Cotta, C. (2012). Automatic evolution of programs for procedural generation of terrains for video games. Soft Computing, 16(11), 1893--1914.

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 9(1), 1.

Henschke, M., Hobbs, D., & Wilkinson, B. (2012). Developing serious games for children with cerebral palsy: case study and pilot trial. In Proceedings of the 24th australian computer-human interaction conference (pp. 212--221).

IJsselsteijn, W., de Kort, Y., & Poels, K. (2008). The game experience questionnaire. Manuscript in preparation.

Iosup, A. (2011). Poggi: generating puzzle instances for online games on grid infrastructures. Concurrency and Computation: Practice and Experience, 23(2), 158--171.

Johnson, D., Wyeth, P., Clark, M., & Watling, C. (2015). Cooperative game play with avatars and agents: Differences in brain activity and the experience of play. In Proceedings of the 33rd annual acm conference on human factors in computing systems (pp. 3721--3730).

Kätsyri, J., Hari, R., Ravaja, N., Nummenmaa, L., et al. (2013). Just watching the game ain't enough: striatal fmri reward responses to successes and failures in a video game during active and vicarious playing. Frontiers Media SA.

Khaled, R., Nelson, M. J., & Barr, P. (2013). Design metaphors for procedural content generation in games. In Proceedings of the sigchi conference on human factors in computing systems (pp. 1509--1518).

Matthews, E. A. (2019). A study of techniques for measuring enjoyment in video games containing procedural generation (Unpublished doctoral dissertation). University of Florida.

Matthews, E. A., & Malloy, B. A. (2012). Incorporating coherent terrain types into story-driven procedural maps. In Meaningful play, 2012: Designing and studying games that matter.

Mekler, E. D., Bopp, J. A., Tuch, A. N., & Opwis, K. (2014). A systematic review of quantitative studies on the enjoyment of digital entertainment games. In Proceedings of the sigchi conference on human factors in computing systems (pp. 927--936).

Mojang. (2011). Minecraft (pc game).

Moore Jr, O. K. (2015). Procedural content generation: Using ai to generate playable content.

Nintendo. (1996-2020). Pokemon (game series).

Nintendo. (1985-2020). Mario (game series).

noise 1.2.2. (2019). noise 1.2.2. Website. Retrieved 2019-06-5, from https://pypi.org/project/noise/

Patrascu, C., & Risi, S. (2016). Artefacts: Minecraft meets collaborative interactive evolution. In Computational intelligence and games (cig), 2016 ieee conference on (pp. 1--8).

Pereira, L. T., Toledo, C., Ferreira, L. N., & Lelis, L. H. (2016). Learning to speed up evolutionary content generation in physics-based puzzle games. In Tools with artificial intelligence (ictai), 2016 ieee 28th international conference on (pp. 901--907).

PyGame. (2020). Pygame 1.9.6. Website. Retrieved 2019-05-20, from https://www.pygame.org/

Pymunk. (2019). Pymunk 5.5.0. Website. Retrieved 2019-06-5, from http://www.pymunk.org/en/latest/

Python. (2020). Python 3.7. Website. Retrieved 2019-05-20, from https://www.python.org/

Saltsman, A. (2009). Canabalt (pc game). Adam Atomic.

SciPy. (2019). Scipy 1.3.0. Website. Retrieved 2019-06-5, from https://www.scipy.org/

Shaker, N., Nicolau, M., Yannakakis, G. N., Togelius, J., & O'neill, M. (2012). Evolving levels for super mario bros using grammatical evolution. In Computational intelligence and games (cig), 2012 ieee conference on (pp. 304--311).

Shaker, N., Togelius, J., Yannakakis, G. N., Weber, B., Shimizu, T., Hashiyama, T., . . . others (2011). The 2010 mario ai championship: Level generation track. IEEE Transactions on Computational Intelligence and AI in Games, 3(4), 332--347.

Takatsuki, Y. (2007). Cost headache for game developers. Website. Retrieved 2007, from news.bbc.co.uk/ 2/hi/business/7151961.stm

Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelbäck, J., Yannakakis, G. N., & Grappiolo, C. (2013). Controllable procedural map generation via multiobjective evolution. Genetic Programming and Evolvable Machines, 14(2), 245--277.

Yu, D., & Hull, A. (2009). Spelunky (pc game).