An Exploration Of Procedural Generation In Video Games

by

Aidan Flaherty

Submitted to the Department of New Media

School of Suny Purchase

in partial fulfillment of the requirements

for the degree of Bachelor of Arts

Purchase College

State University of New York

April 2023

Sponsor: Michel Bell Smith

First Reader: Adam Resnick

Second Reader: Irina Shablinsky

The first person game is a one of the genres of games developed in the early 90s when game developers devised a way to create a three-dimensional perspective for the player, that is to say a player's ability to look up, down and forward. In this way they get a view from the perspective of the player with the player navigating the environment as if he was in it as opposed to controlling the action character within the environment as in a third person or top-down game where your character model or sprite is visible at all times. First person shooters took off in popularity around the mid to late 90s with a dynamic largely action oriented genre styling and the play was essentially centered around the perfection of said action by dealing with enemies swiftly and fundamentally understanding how to use the environment to your advantage, some of these games included Quake, Doom, and Duke Nukem. It was convention for this run and gun format to entail a static environment in which you master the level in question, that is to say, if you die you return to the level and devise a way to traverse it better. As time went on, however, the first person genre expanded into other territories. First person horror games and first person puzzle games also began to take hold in the genre. With these new genres of games, the structure and environment of the games began to change as well. Some games thrived using ever-changing environments and objectives that subverted a sense of mastery by creating an entirely different experience. A game like Quake or Doom mostly has the player fighting their way through maze-like laboratories and underground complexes, finding a key to complete the level and not dying along the way. A game like Half Life which was developed at the end of the 90s sometimes had a stealth level, or in some cases a platforming level, a run and gun level, or a hybrid of all three. Additionally, with game worlds getting bigger and bigger, developers wanted more streamlined efficient ways to develop games and a manual development style did not always apply. If a game was say an open world game with a large forest manually placing every tree, bush, or river, and making every unique model was not really conducive to time strapped development deadlines for games, or a generally deficient way to complete the task. This is where procedural generation began to come into play within game development.

Procedural generation by definition is algorithmically generated content as opposed to manually generated content (Hendrikx, Mark, 3). If you, say, want to make a 3d model of an ocean, you could do it yourself or you could set the parameters to make it happen. One could set the waves to various sizes then set the area, and then command the wave shape to fill the area. One could also find the color of the water from depth to the surface as well. Nowhere in this process do you personally fill in the water or draw the wave, the manual parameters provided by your direct input are simply just set for a computer to create your ocean. However, The nature of good coding necessitates a certain amount of foundational proceduralism. In layman's terms, the program that can do a lot with a little is far superior to the program that can do a lot with a lot. For example, If you wanted to draw three circles to the screen and wrote each circle as a line of code, it could also be done with a for loop which tells the computer that until a variable equals three to keep writing circles to the screen. If you want something akin to maybe a hundred circles this becomes the most obvious way to go about doing this. What if we also want to set a unique location for the circles as opposed to manually setting the parameters for each one, This would be where random functions come in. Which allows you to generate random numbers between given values (Hendrikx, Mark, 3). This is where things begin to get interesting. This is because the use of these random parameters have the potential to resemble the chance based ecosystems that surround us. For example one algorithm describes first characterizing a map by ecotype, like desert, plain, or mountain region and grouping plant species in with their corresponding ecotypes. The map is also further divided into tiles which determine the probability of a certain species spawning within it (Onrust, Benny, 3). If it's a desert region for instance you wouldn't have a pine tree spawn there, instead, you would have a cacti spawn. Then, if you had an oasis area the probability of desert native shrubbery spawned there would be higher than the middle of the desert. This is just one of the reasons why procedural generation is paramount to the development of virtual worlds and is one of the ways that we can begin to perfect these worlds and have them mimic reality.

The argument for procedural generation from a technical standpoint, is the idea that it is less taxing on the CPU (the computer's central processing unit) than a manual algorithm (Hendrikx, Mark, 3). However, at the crux of that sentiment is the even broader idea of decentralization. It could be argued that early computers were a force for centralized power. To begin with, computers were larger than life devices consolidating information in the large rooms of the people that could afford them (Mitchel, Resnick 11). When computers became more efficient and more portable in nature, the information consolidated within the computer and by extension the internet became more and more widespread. Even now there continues to be ideas that are becoming further decentralized in regards to the artificial worlds contained within computers (Mitchel, Resnick 11). Ecological research for instance has influenced much of the ways in which local interactions by certain species have ultimately come to influence larger patterns evolve out of smaller ones. Vegetation, for instance is “self similar or has a structure” (Hendrikx, Mark, 9) plants shapes like “ferns or cauliflower can be generated with fractals” and for more exact proportions “rules may determine the growth of the plant from the initial trunk” (Hendrikx, Mark, 9). This is not purely biological however, and the algorithm known as the L-System or Lindenmayer systems can create recursive fractals for all kinds of places like cities or roads. (Gaisbauer, Werner,4) “They are incredibly useful because they provide a mechanism for keeping track of fractal structures that require complex and multi-faceted production rules .(Gaisbauer, Werner,4) ” and can give us illuminating insight into the broader notion of what naturalism means as our definition can be expanded to natural man made systems as well as nature’s systems. Procedural generation can also extend to the texture or mesh of an object. In a similar manner, regarding the saving of time these meshes could also be generated randomly via perlin noise**.** Perlin noise itself is a pseudo random assortment of pixels that form a sort of noise, hence perlin noise that is so dense in nature that it can aid in generating textures that attain a certain realism. When the parameters are set and the game scene promptly loaded there is a diversity of texture and thus a better mimicry of a natural environment without much CPU power sacrificed (Gaisbauer, Werner, 8).

It can then be argued then that the conclusions derived from natural behaviors of all kinds are transmissible to technological findings if the systems in question find adequate algorithmic translation, in this way the computer can come to mimic this behavior to a great degree of accuracy. (Hendrikx, Mark, 9) In this manner we can come to see procedural generation as an algorithmic strategy that can help us decentralize the notion of the limited and ultimately consolidated artificial world by injecting a sense of chance based naturalism into its lexicon through procedural algorithmic means (Hendrikx, Mark, 2).

Once such algorithms are designed, they can be gathered into coding libraries for implementation in other programs such as video games. One example of this is the boids algorithm which mimics the natural yet chance-based system of flocking birds. The algorithm follows three main tenants all based upon observations of actual flocking birds. The first tenant is separation, which is the idea that a boid (bird-like object) should move away if it is too close to another boid, the second tenant is alignment, which is the ability for one boid to align with its neighboring boids. The third tenant is cohesion, which is the idea that boids move to the center mass of their neighboring boids. (V. Hunter, Adams, 1) Once defined, this algorithm can then be implemented in a program in order to mimic different kinds of flocking behavior depending on how the variables are assigned to the three tenants. One example of border boid algorithm use would be in AntSim which simulates the behavior of ants in a colony (Wong 1). Moreover, algorithms like these can create a sense of cohesion out of what would otherwise be totally random in nature.

Even in the years preceding the so-called golden age of video games, the priority of realism was one that had taken hold of the video game market. As an example video games have currency and in game transactions. Realism in volume and in appearance have come to dominate the industry. Thousands of interconnected areas and highly detailed environments are the norm. Additionally, developer teams of 100 something people are also the norm with regards to the construction of these huge worlds (Hendrikx, Mark, 2). That said many of the procedural algorithms used within these games follow a demand for this sense of realism. The inverse of this, Alienation, or the unnatural in video games is something that in turn have been seldom used. The environments in World Of Warcraft or Call Of Duty as mainstream games are not created for the player to be in a constant state of alienation or subversion but rather in anticipation of a payoff that keeps the player returning.

Eventually game engines like Unity became free and open source game engines like Godot proved to be very beneficial for thinking of a world outside of non-traditional game creation. Much like the cheap availability of early cameras proving themselves to be very useful in the video art world for exploring the potential that the medium had for concepts that did not prioritize pleasure or commodification. Any one person could essentially download the Unity engine, make an account and create a game. As long as you aren't making money off your game the software itself is totally free. This opens up the floodgates of the kinds of people using the program and the kinds of goals that they would have if they chose to go ahead with a serious project. Hence previous structures of the medium could be deconstructed or destroyed all together (Mitchel, Resnick 8).

In the video game world with the advent of user created games enabled by platforms like Unity, one of the questions that began to become deconstructed was the question of whether the dynamic of mastery was needed. In place of mastery, what if the gameplay’s theme was perpetual helplessness from beginning to end. For instance an enemy could not directly attack, but tease the player and attack when his guard is down. Not only that, but the player would also be at a constant disadvantage through the game. This is where procedural generation began to bleed into video games at large. The idea of being perpetually on edge or the ruleset that is never fully known is something that coincides very closely with the horror genre and this is where procedural algorithms began to make sense. Games like Slender The 8 Pages give a group of designated areas for the enemy, or in this case Slender Man to appear with a randomized chance as to which location the algorithm chooses so as to always give the player the feeling that no matter how close they are to the goal of collecting the 8 pages they are never any less helpless. Another game called Amnesia would start out normally before gradually getting the player lost by algorithmically generating passages that, because of their randomization, became more liminal and warped. This effectively creates a pit in the players gut that grows deeper and deeper as they continue to play.

One of the new paradigms that these games opened up was the incorporation of continuous continuous performance which is defined by the way in which we measure someone's performance of a task over an ever changing paradigm (Münscher, Johann-Christoph,1). This means that the test is getting harder and harder in response to the players actions. A test like this is similar to a horror game like Slender which determines the areas the Slender enemy will appear based on where the player is. Older versions of this test called the Stroop test are only measuring attention through a limited and controlled paradigm in which the difficulty does not take the user into account. One that does not reflect the realities of attention, the continuous attention involved in ever changing or continuous tasks in which many conditions reframe and reshape themselves over the course of the test. Much like in a game like Slender or

Amnesia there is no point at which the player or test taker can accurately anticipate a fixed point at which something will happen. It is the goal of both the test and a horror game to demand this continuous attention because they are essentially measuring a similar thing. The CMT is measuring continuous attention for a disability like ADHD and a horror game like Amnesia or Slender is trying to put you on edge such that you are attempting to be wary of every sound or shadow that could possibly indicate danger. (Münscher, Johann-Christoph,2) These tests were also not necessarily computerized in nature meaning that they could embody a computerized interface yet they could be done without a computer entirely.

The paradigm of video games works in a similar fashion, that is to say that their rulesets can mostly be replicated without a computer. The procedural appearance of a video game character cannot be truly randomized in the context of a board game for example, games that do not require continuous attention can often be transmissible to the format, like say Warhammer for example. Similarly, the CMT test, which measures continuous performance, requires that the participant is trying to constantly minimize the distance of a target object that is in constant flux up and down the screen. The participant is told to move the character with a sliding panel. Thus in order to complete the task the participant must essentially remain constantly attentive. (Münscher, Johann-Christoph,2) A game like Slender The 8 Pages works within a similar paradigm. Although perhaps despite the goals being similar in nature, the methodology of Slender is perhaps the inverse of the CMT. That is to say that the participant of the CMT is always moving closer to the target and has to pay continuous attention to make sure this occurs to the best of his ability. In Slender the participant is doing his best to move away from the target continuously. In order to do this, both the player and the enemy have to be constantly aware of the target, and in the case of Slender, they have to be aware of any correlating factors that regard the target. In the game, particularly, there are many things that correlate to the possibility that the enemy or Slender as he is called might appear here or there. You might pick up one of the 8 pages that you are supposed to collect and have to check your peripherals for the static which appears when the Slender character is near your person. You might find a new area far from the part of the area that you have familiarized yourself with, and you might say to yourself yeah the Slender Man could appear here. And because he appears there enough times that you might be wary of these factors. It can be deduced then, that the player must utilize continuous spatial attention which is keeping them perpetually on edge on every front. In a way it can be regarded as a 2.0 of the CMT in terms of the scope that the attention demands. The CMT test and aforementioned has the player hit a target on a sliding scale with the CMT target responding procedurally and in accordance to the player’s accuracy (Münscher, Johann-Christoph,2).

One thing to note about a game like Slender is its big and overwhelming it can feel, one of the core tenets of the game is how vast it is and how easy it is to get lost. This harkens back to one of the original reasons for using procedural generation which is obviously to be less taxing on a user's computer and thereby increase their market base. One of the other reasons for this is more conceptual as it can aid the game in creating the illusion of having a space that calls for a certain vastness to the area in which the game plays out (Gaisbauer, Werner, 4) . The result is having a large yet populated terrain for the player to traverse that is essentially executed in one or two lines of code.

Many of these vast game environments like the ones in Slender The 8 Pages and the like are meant for areas that would have the most overhead if they were to be made with bigger teams of developers. These environments would largely include cities and forests or widespread dungeons. Mimicry of naturalism does not come purely from the process or randomization itself but rather parameters through which to randomize. Textures and placement all serve to create a good flow to the game or at the very least a sense of naturalism. Other things like behavior tend to make a lot of sense for wildlife or idling hostile characters as you can give them parameters with which to roam and with those parameters give the player an unpredictable stealth session or simply have the cpu power that it would take to physically write out the action (Hendrikx, Mark, 6).

One thing that also makes this dynamic interesting is the introduction to the uncanny that comes through this notion of unpredictability or the inklings of unnaturalness that come through when a level is both vast but also nauseatingly intricate. It is here that we essentially begin to see a certain kind of effect that is wholly the byproduct of procedural generation. The feeling that human influence is limited and thus the space is lacking in both the natural and (by extension) the rational.

Many of the games that utilize this strategy aim for a more liminal or maybe unnatural approach to the gameplay in question. One of the games that most commonly uses this technique is the dungeon crawler, a genre of game which entails a character traversing a vast labyrinth. As an example, in the procedurally generated dungeon crawler, The Binding of Isaac, there is essentially an invisible grid through which various rooms are spawned randomly within said grid. The very concept of a vast dungeon, especially one with a randomized assorted arrangement of features is essentially uncanny or unnatural and it can feel overwhelming to traverse levels. (Boris, 1).

Games like The Binding Of Isaac however try to strike up a balance between order and randomness by trying to keep a certain amount of constants to the environment in question. They will generally do this by having certain set areas where game objects appear, along with templates within each room for a variety of objects to appear in places that make sense. Many dungeon crawler games such as Spelunky use this strategy to ensure that the game still makes enough sense for the player to be able to meaningfully interact with the world while still creating a sense of liminality to it (Boris, 1).

The functionality of these level designs is mostly focused on the blind spots that exist within our current understanding of how to make procedurally generated levels. Many of the blind spots revolve around the notion of how one can essentially make an environment that feels natural. This is why procedural methods like the ecotype method of determining species for environments as well as the probability of them growing in a certain spot is important.

Methods like ecotype determination increases the likelihood of an essentially canny simulation within the world and makes it so that the world is both an immersive artificial world while also being essentially natural. One problem that is even bigger with regards to the random environment would be the problem of creating objects that are generated randomly in both size and appearance. While the aforementioned use of perlin noise is able to aid in the creation of randomized material to a certain extent does not completely solve the problem when taking into account the large number of polygons and surfaces that have become industry standard within the modern game space (Onrust, Benny, 2). The aforementioned L-systems that can generate recursive fractals based on the plant stumps for realistic looking plants, roads, or cities are largely responsible for making a canny world. Uncanniness can come into play however, in the setting of more radical parameters are set. Perhaps we essentially make up the rules for how a plant should grow, making it grow spastically in every direction based on one kind of stem or stump and maybe have another plant that does the inverse. It is possible too that you can make this plant a fantastical species unique to the game as well. It is in this way that the procedural systems used to mimic naturalism can also mimic the fantastical (V. Hunter, Adams, 1). As an example, the boids algorithm was used in swarm robotics where it was not necessarily naturalism that they were after, but instead were trying to program small robots to do specific tasks as a flock or colony. In their assignment of these tasks they are not mimicking a species per say but rather becoming an entity that can complete the task (Wong 1).

With the horror game genre this becomes more pronounced. One might ask, how uncanny should I make it, how frequent or unpredictable do I make this character in order to establish an environment in which the player feels as though they should venture in a certain direction to discover secrets or power up? Or when should I make the player feel as though they should put blinders on and carry onward? The best games are the ones on all of these fronts. This is made doubly hard when procedural generation is brought into the picture. One might ask if there are too many creatures spawning in this area will it deter the player from going there? Additionally, how then can the player progress in this map with the game constantly getting harder and throwing surprises at the player? The most important question to ask then is how exactly do we create the aforementioned environment of continuous attention in a virtual world dominated by procedural techniques? We fundamentally understand that at this point the technique itself often results in the uncanny. One important distinction is establishing the kinds of fear that you might want to induce in a player. When taking this into consideration there are two types of fear that one might want to establish. The first are non cognitive reactions and the second is cognitive. The non-cognitive reaction is referred to more commonly as the jumpscare or the ability for a scare to make someone do something without thinking due to the fact that the result is a sudden jolt to one's inhibition. In the surveys done on the subject this is more common in children and less common in adults. The strategy that is more common in adults is the non cognitive strategy (Tammy Lin, 3).

One will rationalize or try to make sense of the environment or try to go about the environment carefully and resourcefully in response to the potential threats at hand. One of the findings that the survey conducted by Tammy Lin for National Cheng-Chi University that wanted to determine “what causes audiences’

fright in the VR horror context (Tammy Lin, 2)” and “how audiences with different individual traits cope with these mediated threats indicated, (Tammy Lin, 2)” was that in these older adults the non cognitive responses were a sort of phenomenology of cognitive responses. That is to say that these people were slowly becoming aware of the indicators of the potential threats at hand and as they continue to realize which indicators are associated with potential threats they begin to get better at avoiding them or at least finding the means to deal with them accordingly so that the non cognitive response never occurs in the first place (Tammy Lin, 3). In turn we begin to get a good idea of the things that indicate a certain safety and in this way, can begin to make accurate predictions of which parameters to set within the procedural environment. One of the further questions to ask is the question of prediction regarding how we ensure that we understand at all times that the players' sensibilities when playing are ones that are in full alignment with a CMT like experience. One in which, much like the CMT, the game responds to our input and is thus as aware of us as we are thereby making unique decisions for what to throw at the player every playthrough. (Münscher, Johann-Christoph,1).

To conclude, these are questions that can only be meaningfully answered by better understanding the checks and balances of procedural generation and continuing to look at the study of natural systems as a way to continue to optimize the artificial space by tearing down the delineations between the artificial and the natural. Not just in appearance and more generalized ecology of game environments, but also in the ways in which we interact with the space. This can come through studying spatial awareness and attention as shown with the cmt tests. In this way we can use procedural generation to carve out new frontiers in virtual space.

This is my primary argument for my decision to use procedural generation within the game I created. I started out my project with an extremely limited knowledge of the Unity platform and game building more generally. I had programming knowledge, but had generally only worked on the front end meaning that most of my programs dealt with aesthetic features with little to no interactivity. I came to Unity after learning the fundamentals of backend programming and a semester in which I had focused exclusively on back end fundamentals by working with the programming language Java. This allowed me to create dynamic algorithms for programs that could be interacted with and problem solve with relative ease, which aided in my general ability to create a game within an optimal timeframe. Algorithms for games and especially first person games however were a bit distinct from the backend coding that I had done previously. Thankfully, the Unity library was able to give me a very introduction to the algorithms required to create dynamic worldbuilding both with regards to the aspects of the world that were procedurally generated and the ones that were not.

One of the main ideas that I had from the moment that I began to create the game itself was that through procedural generation, certain elements in the level could either put the player at either an advantage or disadvantage. Before we understand how the game elements and their procedural aspects both aid and debilitate the player in their implementation however, it is important to understand how exactly my game works. The game works as an environment that, while, is similar in appearance, fundamentally shifts from level to level. The first three elements that were created within the world were trees and rocks and then the forest floor itself. The forest floor created a pen-like structure with walls blocking the player from falling off of the edge of the environment. Within the forest floor, rocks are given specialized sizes and trees are given specialized placements within the map. The first procedural algorithm used in the game is utilized for one of the more common uses of procedural generation; to create a simulation of sporadic growth. This can speed up the otherwise tedious development of forest creation in a video game setting. This sporadic growth is regenerated for every level in the game which are described in game as nights (since the entire game takes place in the dark).

My use of sporadic growth was not exclusive to the trees and to the rocks within the environment, but also everything else like enemies and orbs which can be picked up to use as projectiles. The earliest stage of playtesting made certain it was necessary that these elements always spawned in a quantity that would ensure that every level had enough checks and balances to be playable. It had to both be different and the same in a balance that keeps the game enjoyable: never rigged in the enemy or players favor. The player couldn't be given too many orbs or too many enemies for example, likewise, too many trees and rocks could make the environment too hard to navigate for the player. One variable in each spawn script was the guiding parameter that essentially kept the game balanced. Another recursive algorithm that keeps the game balanced, was one where every element had a loosely set box to spawn inside, one that nothing else could spawn inside. This, however, is not a perfect system, which leads me to another leading idea for my game: the inclusion of the uncanny. I wanted within the game, a slightly off or cartoonishly warped environment that would be congruent with the other uncanny or cursed aspects of the game.

The uncanniness in the game is a product of a practice that I employ with any new tool that I come across and decide to use. That rule is that if there is some glitch or abnormality I lean into it and let it lead me in whatever aesthetic (or in this case, gameplay direction) that it may.the first example of this in my game was me letting environmental elements overlap when they spawn but giving enough space to each object that it does not result in total chaos. Then other necessary canny elements were also added, like your actual character and its respective weapons being able to collide with any object and interact with it accordingly. In fact, this uncanniness is one of the reasons I used procedural generation within the game itself due to its blind spots that add this sense of uncanniness, despite being unsure of the way it was going to manifest itself within development.

The gameplay loop consists of defeating all of the enemies within the environment and then returning the crudely constructed shack that serves as the game's beginning point. Through the level an arrow will point at the nearest enemy area, if you have taken out all of the enemies the arrow will point back at the house so that you can move onto the next night. Small attributes like these allow for some semblance of a guiding light in an environment that is ostensibly (despite a few constant factors) a total mystery to the player each time it is created. It allows for some general encouragement to move forward when confronted with a level that, aside from your headlamp and distant lights from orbs or enemy areas, is totally dark.

Enemy areas are another parameter through which the randomness is curtailed. Enemies are divided up into three categories, plant, animal, and humanoid. Their spawn boxes are indicated by flesh trees that are deadly to the player. To indicate plant enemies, three of these trees are arranged in a square around the plant enemy, slightly wider in range for the first enemy and smaller for the second. The arrangement is always the same so that there is always in essence some indicator of the enemy that you are dealing with. I would refer to these flesh trees as a sort of enemy subclass called deterrents. Not hostile in that they actively pursue you, but they can take away health and in the case of the flesh plants move up and down and back and forth when alerted to your presence. That said, one of the ways I chose to provide balance within the random nature of the game was to provide some enemies with an explicit motion or ruleset that is totally predictable. The category of deterrent objects explores a perfect duality between the nature of an enemy type that features an explicit ruleset and one that is relatively explicit and exhibits behaviors that cannot be predicted in the same way. Another enemy is the red orbs, it should become clear to the player that a blue orb has a good association and if you pick one up you have ammo and a health point if you are below 5 health. The reverse is true for a red orb which can be seen at a distance glowing over an enemies spawn box. These orbs move using a recursive algorithm similar to the previously described spawning algorithm as well as the collider algorithm. However another script always points the orb at the spawn object itself. So despite its randomness it always moves around the same vicinity which is the enemies spawn box. Its unpredictable and erratic movements should incentivize the player to get a quick hit in and get out fast in a video game player strategy known as strafing.

The first of the enemies I created were the plant enemies, these enemies are relatively passive and the easiest in the game to deal with. That said, the plant enemies were there from the earliest stage of development and determined the way enemies would spawn along with the way they would get hit and die. A hitbox would be featured as the place the object would take hits and once it dies a particle system deparents and plays to indicate the death of the object. The first enemy consists of three trees with their appearance or texture coming from an image of red grapes. Similarly, the tree deterrents which use an image of small pox for their texture to create the flesh tree look. I used many different images to dictate object texture and appearance. Many came from obvious sources like a grass image for the ground and bark for a tree but some were not so obvious. I used pictures of eyes infected with parasites for enemy eyes or images of syphilitic for enemy mouths. These enemies spawn at the camera rotation and shoot in that exact spot hoping to have the angle of surprise and get a hit on the player. Second comes the turret enemy turns to face the player hoping to always get a hit on you at all times.

Plant enemies are mostly meant to have a predictable attack pattern and are meant to have one surprise up their sleeve when attacking the player. For instance the unmoving plant enemy appears and faces the player as it steps into its area, but does not readjust as the player moves. In contrast many of the procedural techniques used resided in higher level humanoid and animal enemies who would dodge unpredictably or move faster and in different directions when it knew it was being attacked. Additionally, some of these enemies could shoot orbs in the direction of the player with the timing at which they shoot and the projectiles reach being determined by procedural means. The first enemy I created to embody these actions was the bull creature. I called it the bull due to the fact that it would advance towards the player and redirect itself at a time interval which is decided procedurally. If the player is too close it lunges forward and if the player is behind it it lunges backward. Like all humanoid enemies in the game it is free floating and able to move wherever it wants on the map. It has two arms that drain player health and also shoots the aforementioned randomized projectiles. The images used for the enemies appearance came from syphilitic mouths as well as a picture of pink eye, for its arms I used an image of skin infected with smallpox. For more advanced objects I used Pro-Builder which is an add on to the Unity software that allows for light 3d modeling. I did these for more complicated shapes that unities stock shapes like sphere square or triangle could not justify. So for all enemies I used incredibly simple shapes with very real images, these are essentially two juxtaposed elements that are perplexing to look at when combined. The bull consists of two tree objects (a stem-like shape that comes with Unity stock objects) for arms then a stock capsule shape for the mouth and two spheres for eyes.

There were other aspects of the game that essentially got away with a total randomness in all respects as their job was to define the games overall atmosphere as uncanny or cursed. The sound in the game was a primary example of this and it was implemented by using a spawner script similar to the rest but with no real definition of the radius through which to spawn. Additionally, Every box consisted of a 10 second audio file which would play when the player walked into it and all of the audio clips were meant to inform each other and form a broader whole. If the player is to move around the world the player will begin to hit boxes and 30 second stems or audio clips that inform this four minute low drone begin to play as you run into them. This generally only goes for background sounds as combat stems and enemy sounds are played when they are alerted. As for the predictive actions that make this algorithm work, this falls on the player himself. If the player is moving fast he will begin to hit many boxes causing the stems to form a more chaotic and possibly erratic variant of the piece but if he moves slowly the opposite occurs. Due to the fact that before programming my main interest was making music with programming being a close second, I was looking for new contexts with which my auditory works could be explored and I can confidently say that I found a really effective way to create effective worldbuilding through this method of interactive sound design. That said, at the outset the entirety of the soundtrack was mixed in the same digital audio workstation session and in practice, meant to form an oozing wall and culminates in a sort of erratic melodic sequence upon alerting an enemy which can be described as a climax of sorts.

Many of the sounds themselves, like a lot of things in this game, were made from scratch. I worked with raw sine waves shaping them in various programs like Emission Control, Reaper, and finally Ableton until a subtle but chaotic dark ambient/electronic tack began to emerge. I did a similar thing for enemy and player sounds but focused more on raw data and tongue in cheek plunderphonics. I stole the player hit soundbite from a Slayer song with the vocals isolated, humanoid enemy sounds came from a James Brown interview with their hit sounds coming from the infamous down with the sickness vocal stab from notorious Walmart rock band Disturbed. Keep in mind that in many ways they were tweaked in Ableton beyond recognition to the point that the samples may as well have been creative commons. One thing that i found when doing most of the sound design for the game was the idea that there were sounds that i could definitely record or find on royalty free sites instead of doing what i did but the fact that a certain sound can approximate another sound without it being a full bullseye as to the accuracy it adds to the cursed or uncanny thesis of the project and reminds the player that something is off. This is because despite looking a little bit like a human, plant, or animal, they most certainly are not. Because of this, using a human, plant, or animal sound would not fully embody the essence of these creatures making more straightforward sounds disrupt the cohesion. My favorite example of this is the fluttering of passive flying enemies who emit a fluttering sound made with heavily filtered white noise when you are close. In one instance it sounds like fluttering but in another it sounds like something is wrong with your computer and thereby puts the player on edge. Despite this, I made sure that these sounds were fairly unassuming and relatively smooth on the ear and mixed so as to be uneasy but not jarring. This method is another example of exploiting a blind spot.

In the final stretch of development, however, I zeroed in on the blind spots that I could not fix. One of these was the general sense of direction within the game. I sensed that the player would not know where to go. In solving this issue of direction through pointing the arrow at enemy spawn areas I was able to complete the game itself. If the arrow could point at an object, and assign when a condition is met (like walking into the enemy area the arrow is pointing at then pointing to another area) then enemies can do the same. The second humanoid enemy essentially utilized this method of object finding and redirecting by finding a tree (an actual tree model not the unity stock object) to move in the direction of when hit. The that the creature moves to is totally random and in this way it's unpredictable as to where the entity will move or how far it will move when hit. The enemy consists of a tree object and capsule object and a sphere representing phallus and body and eye respectively. The creature is tank-like and moves in the direction of a random tree as previously stated. Once there it shambles at random around the desired object until the player strikes it again. Out of the phallus like object smaller enemies spawn, similar to the bull it two tree objects for arms that lower player health when hit and has a body two eyes and mouth all made out of spheres and using the same series of medical photos of pink eye, mouth infections, smallpox and the like. Unlike the creature they spawned from these enemies are fast and agile and like all of the humanoid enemies, are free floating and chase you at all times. However, to balance things out they only take a few hits to take out.

It is in this way that each level begins to retain a certain character and features a procedural album that creates order as opposed to one that is in essence random in nature with a sense of order being provided mostly by a separate script entirely. In a similar way, this is how the passive health giving enemies work in the game, many people who played it, classified the game as permanent death adjacent but if one is wounded he can look to the relatively passive animal enemies for health. Animal enemies are the third kind of enemy within the game and they primarily make use of finding a random tree. However once they find it, they do not wait to be hit, they just move to another tree once they reach it. In this way, they tend to be in the player's vicinity more often than not and will attack the player when they are in its line of sight. These enemies' appearance circumvents the medical photos and instead tends to opt for photos of animal fur and up close scans of bug eyes specifically of the spider and fly variant. Because of this, one enemy does look almost spider-like and the other almost fly-like with their wings represented as tree objects with the image of smallpox as the image on the object for added uncanniness. If the player steps into the fly's vicinity, the fly will begin to follow the player and shoot red orbs in the player's direction. The wings on both the fly and the spider are deadly and running into them is another way to get killed. The spider enemy is similar in functionality but normally finds a tree and rotates around it before moving to another tree. If the player steps into its vicinity the spider accelerates and two flesh tree objects on the front of its person will snap at the player and drain the player's health when hit. The fly enemy will never stop following the player but is generally slow but the spider will stop following you if you get out of its hitbox for long enough. The benefit of actually engaging with these enemies is that as previously stated, hitting them will give you a health point that you will very much need because you only get 5 at a time. These enemies are always in the game; they do not appear when you are in an enemy area and can thus be avoided if the player is careful and does not need health. The health mechanic however, balances out how unfair the game can be healthwise, you only have 5 health but agility, not hitting enemy areas, and trying to stay out of passive enemies lines of sight will allow the player to make it to another night.

At the end of development the game became a sort of crude and unforgiving cartoonishly uncanny first attempt at a game. Other than the aforementioned orbs the player only has a pickaxe with which to defend himself. In addition there are bigger areas with no trees off to the sides of the map where it is easier to avoid taking damage. This allowed for moments of intense action to combine themselves with a stealthier horror experience in the gameplay which I found to be somewhat unique. Additionally, all objects can essentially float through other objects except for the player, making the whole experience odd and dreamlike and adding to the uncanniness of the experience as a whole. The appearance of the game was at once very real due to the image used on an object coming from real life, but also cartoonish in that every object and character in the game was constructed with such simple shapes. In retrospect I could have been more meticulous in striking up a balance between player and enemy, however it being my first game I dove into the dynamic qualities of each enemy headfirst possibly giving myself too little of a time to playtest with the difficulty curve in mind. In addition, I think I understand more than ever why so many games avoid a generative approach to development. I own a mediocre laptop, while my game is not very graphically taxing the game environments still took around a minute to load. Because the difficulty curve was high, the load time became hard to justify, yet it was necessary to generate the unique and somewhat natural looking environment the game relied on. I attributed this to the fact that a lot of these random algorithms were CPU-intensive and I should have put more effort into making them less taxing. I also think that more space in the pen or forest floor would have aided navigation a lot in addition to curtailing the speed at which some of the sprites sped at the player. Lastly, I would like to have looked more into why this game has a lot of trouble being played in full screen and tried to fix that before I did the final build. I am happy, however, with the speed at which it currently runs and since it is quite arcade-like, running it like a dashboard widget makes a lot of sense. My research really aided my understanding of many of the checks and balances that i would have to undertake with making a procedurally generated game and things would have been a lot slower regarding my general understanding of creating a sort of self governing ecosystem had i not dug into the guts of the pros and cons of procedural generation inside and outside of the gaming world. I hope to continue going in this direction in future works and make better and better self governing systems as I move along. The link to my itch.io page can be found below and the game itself and any updated versions will be free to download.

the jank forest survival game by 0164 (itch.io)

List of games mentioned:	software mentioned
Slender The 8 Pages	Unity Engine
Amnesia The Dark Descent	Emission Control
Quake	Ableton
Doom	Audacity
The Binding Of Isaac	Reaper
Spelunky	

Works cited:

Münscher, Johann-Christoph, et al. “The Continuous Matching Task (CMT) - Real-Time Procedural Stimulus Generation for Adaptive Testing of Attention.” Applied Neuropsychology. Adult, vol. ahead-of-print, no. ahead-of-print, 2021, pp. 1–14, doi:10.1080/23279095.2021.1969399.

Hendrikx, Mark, et al. “Procedural Content Generation for Games: A Survey.” ACM Transactions on Multimedia Computing Communications and Applications, vol. 9, no. 1, 2013, pp. 1–22, doi:10.1145/2422956.2422957.

Gaisbauer, Werner, and Helmut Hlavacs. “Procedural Attack! Procedural Generation for Populated Virtual Cities: A Survey.” International Journal of Serious Games, vol. 4, no. 2, 2017, doi:10.17083/ijsg.v4i2.161.

Bernardi, Alessio, et al. “Procedural Generation of Materials for Real-Time Rendering.” Multimedia Tools and Applications, vol. 80, no. 9, 2021, pp. 12969–90, doi:10.1007/s11042-020-09141-9.

Silva, Rafael C. E., et al. “Procedural Generation of 3D Maps With Snappable Meshes.” IEEE Access, vol. 10, 2022, pp. 43093–111, doi:10.1109/ACCESS.2022.3168832.

Zafar, Adeel, et al. “Search-Based Procedural Content Generation for GVG-LG.” Applied Soft Computing, vol. 86, 2020, p. 105909–, doi:10.1016/j.asoc.2019.105909.

Bulgakova, Oleksandra S., et al. “Algorithmic Modifications in Procedural Generation Systems.” Control Systems and Computers (Online), no. 3 (281), 2019, pp. 52–59, doi:10.15407/csc.2019.03.052.

Onrust, Benny, et al. “Ecologically Sound Procedural Generation of Natural Environments.” International Journal of Computer Games Technology, vol. 2017, 2017, pp. 1–17, doi:10.1155/2017/7057141.

Liu, Jialin, et al. “Deep Learning for Procedural Content Generation.” Neural Computing & Applications, vol. 33, no. 1, 2021, pp. 19–37, doi:10.1007/s00521-020-05383-8.

Author links open overlay panelJih-Hsuan Tammy Lin, et al. “Fear in Virtual Reality (VR): Fear Elements, Coping Reactions, Immediate and next-Day Fright Responses toward a Survival Horror Zombie Virtual Reality Game.” *Computers in Human Behavior*, Pergamon, 27 Feb. 2017, https://www.sciencedirect.com/science/article/pii/S0747563217301383?via%3Dihub.

Author links open overlay panelJih-Hsuan Tammy Lin, et al. “Fear in Virtual Reality (VR): Fear Elements, Coping Reactions, Immediate and next-Day Fright Responses toward a Survival Horror Zombie Virtual Reality Game.” *Computers in Human Behavior*, Pergamon, 27 Feb. 2017, www.sciencedirect.com/science/article/pii/S0747563217301383?via%3Dihub.

V. Hunter Adams. “Boids Algorithm - Augmented for Distributed Consensus¶.” Boids-Algorithm, vanhunteradams.com/Pico/Animal_Movement/Boids-algorithm.html.

MITCHEL RESNICK. Turtles, Termites, and Traffic Jams : Explorations in Massively Parallel Microworlds. Cambridge, Mass: A Bradford Book, 1994. ISBN 9780262680936. Disponível em: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1998&site=ehost-live&scope=site. Acesso em: 10 maio. 2023.

Boris, |. “Dungeon Generation in Enter the Gungeon.” *BorisTheBrave.Com*, 31 July 2022, www.boristhebrave.com/2019/07/28/dungeon-generation-in-enter-the-gungeon/.

Boris, |. “Dungeon Generation in Binding of Isaac.” *BorisTheBrave.Com*, 12 Nov. 2021, www.boristhebrave.com/2020/09/12/dungeon-generation-in-binding-of-isaac/.

Wong. *Boids*, cs.stanford.edu/people/eroberts/courses/soco/projects/2008-09/modeling-natural-systems/boids.html. Accessed 11 May 2023.