PROCEDURAL CONTENT GENERATION IN GAMING VIA EVOLUTIONARY ALGORITHMS

Serdar Cetiner

Dissertation

presented as partial requirement for obtaining the Master Degree Program in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

PROCEDURAL CONTENT GENERATION IN GAMING VIA EVOLUTIONARY ALGORITHMS

by

Serdar Çetiner

Dissertation / Project Work / Internship report presented as partial requirement for obtaining the Master's degree in Advanced Analytics, with a Specialization in Data Science

Advisor: Professor Mauro Castelli

February 2023

STATEMENT OF INTEGRITY

Serdar Çetiner

February 28, 2023

ACKNOWLEDGEMENTS

This thesis has been an amazing opportunity where I was able to explore both my affection for gaming and my passion for Data Science. The work done would not be possible without Professor Mauro Castelli, who not only provided with me the guidance I needed but also gave me the autonomy to follow the uncommon domains to conduct the research.

I would like to thank my friend and colleague Oguz Kokes, who has been with me from the beginning of this journey and whose help, especially on the conceptual design, has been instrumental in the creation of this work.

Lastly, I express my gratitude to my family; my parents İhsan and Meryem Çetiner for their unwavering love and support in every endeavour I had in my life, and especially my brother Faruk Çetiner, who has been a constant role model, a mentor, and a good friend for me. This work would not be possible without them.

ABSTRACT

The aim of this thesis is to investigate the possibility of creating content using the Genetic Algorithms. To this end a simple system of interconnected algorithms were developed using concepts from Role Playing Games, specifically Dungeons and Dragons to create game content as characters, quests, and encounters.

To be able to produce context, subsystems of map, character, quest, and encounter generators were created. These systems or engines not only define the game space to be populated, but they also provide each other input to create maps, quests, locations, animals, and events that are sensible and coherent.

Randomness of the generation was essential as such a variety of noise maps and random number generation were added to every engine in the system. Layered or singular noise maps allowed for logical assumptions to be made, like seeing camels in a location with no rain and high temperatures. With the base truth coming from a random noise map such as danger, civilisation, faction etc., each system built on top of each other can get more complex.

There are several Genetic Algorithms with custom operators within the system. These algorithms take their inputs and individuals from the respective engines and tie them all to each other through their physical coordinates in the gaming space. The most impactful part of these algorithms is the Fitness Functions defined with concepts from literature or CGI.

The proposed system can populate a game space with elements of desired attributes given the constraints. The output produced consists of coherently tied story beats with some attributes already set. Even in this simple level, this can allow not only game designers but anyone who wants to build any kind of fictional work.

KEYWORDS

Genetic Operators; Optimization; Genetic Algorithms; Procedural Generation; Map Generation; Quest Generation; Character Generation; Noise Maps

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

NPC	Non-Player Character
RPG	Role-Playing Game
MMORPG	Massive Multiplayer Online Role-Playing Game
RTS	Real-Time Strategy
GA	Genetic Algorithm
DnD	Dungeons and Dragons
PMX	Partially Mapped Crossover

1. INTRODUCTION

1.1. Background Information

In the beginning of the millennia, video games were quite like other entertainment mediums in their linearity. Players would engage with a pre-defined narrative with a clear structure of beginning, middle and end. Players would pay a certain fee to gain access to the game and that would be the end of the transaction. (Singer & D'Angelo, 2021)

In the recent years, our consumption of content and entertainment has grown exponentially. (Koetsier, 2020). This phenomenon is more apparent in the gaming sector as games -especially open-world video games and mobile games- are changing from the up-front monetization model to ongoing-based revenue models. (The Evolution of Business Models in the Video-game Industry, n.d.)

In these models more time spent in the game, the more lucrative it becomes with micro-transactions and add-on content. Procedural generation caters exactly to this need, as quality content created at a low cost is the main way to increase revenues.

On the technical level, procedural generation is a method of creating data algorithmically through computed randomness offset by human-generated rules and assets. Various elements of the games as textures, maps, NPC's, weather, and assets have been created with these methods. (Ebert et al., 2002)

Genetic algorithms are a series of computer optimization techniques that mimics the way natural selection works. These include mutation (changing a gene in an organism's DNA), crossover (exchanging parts of two-parent organisms' genomes) and selection (keeping the best offspring). In these algorithms, a solution's "goodness" is determined by a mathematical function. The ideal answer to the problem is called the "fitness function" and can be used as a measure of how good or bad each solution might be. (Goldberg, 1989) The fitness function is human generated, meaning that "fitness" can be defined as anything if it can be mathematically represented.

This thesis explores the possibility of creating content by the means of genetic algorithms. The aim is to utilize gradient noise as the bases of randomness and human-created fitness functions to create coherency and context on different views and systems to a potential game as a content blueprint.

1.2. Procedural Generation

Ebert defines procedural technique as code segments or algorithms that specify some characteristic of a computer-generated model or effect. He gives an example of procedurally generated texture for a marble surface that does not use a scanned image to define the values. (2002)

In gaming, Tanya and Shorts talk about how they find procedurally generated content feels less controlled than any human-made content as the world will be more random and realistic. They also state that procedurally generated content might ruin the game design and cost more if it is not properly integrated into the core mechanics of a given game. (Short & Adams, 2017)

1.3. Genetic Algorithms

In computer science and operations research, a genetic algorithm is a computational intelligence algorithm that mimics the way real-world organisms evolve by selectively breeding favourable traits. Genetic algorithms use mechanisms like those found in nature such as mutation, crossover, and selection—to create high-quality solutions to optimization problems. (Holland, 1992b)

A generic genetic algorithm includes the following steps: initialization, selection, reproduction, and replacement. In initialisation a suitable encoding scheme is used to create an initial population of individuals; then selection takes place where in the best individuals are chosen based on a certain criterion by considering their fitness values; next comes reproduction where each individual produces a new generation using its own encoding scheme in order to create offspring having better characteristics than that parent and finally replacement step defines which operation should be done with non-fittest ones among them that do not fit for survival or improvement purpose. (Kora & Yadlapalli, 2017)

1.3.1. Fitness function

Genetic algorithms start with randomly initialized populations of potential solutions. To steer the algorithm towards desirable solutions a measure of performance is required. The objective function, or often called the "Fitness Function" represent the problem mathematically as a single figure of merit. Fitness function is always problem dependent. (Lambora et al., 2019)

1.3.2. Genetic operators

While genetic algorithms are powerful search strategies that can be applied to a wide range of problems, it's often difficult to design an optimal set of operators for any given problem. This is because the same operator might work well on one problem but not on another. (Yao, 1993)

Genetic operators are applied during the iterative process of modifying the genomes of best "individuals" for each iteration, also called "generation".

1.3.2.1. Selection

During the successive generations, the best individuals of populations are selected to breed the new generation. In essence, the individuals with higher fitness are more likely to be selected for breeding.

Depending on the problem there are many selection methods. Roulette Wheel Selection method sets an individual's probability of selection proportionately to their fitness. A similar method Rank Selection sets the probability of selection depending on their ranks which makes it useful when fitness values of individuals are close to one another.

1.3.2.2. Crossover

Inspired from nature, Crossover operators in Genetic Algorithms are the main point of "genome transference" between parents and offspring. (Holland, 1992b) Crossover algorithms are specifically dependent on the application as they work with the gene encoding of the parents. As such, existing crossover operators were modified specifically for this application for the evolving process of maps, quests, and quest lines.

Umbarkar and Sheth, in their extensive review of Crossover operators, split them into three buckets: standard, binary, and problem dependent. Standard crossover operators work with numeric representations of genomes and uses mathematical or statistical methods to combine the parents to create offspring. Binary crossover operators use binary encoded parent genomes and utilise similar methods as standards but can treat genomes as vectors. The operators created for this thesis falls under the third bucket.

Several existing operators were tested for the use and specifically modified versions of Partially Mapped Crossover (PMX) were proved effective as the main important factor was the location for each individual for this application. PMX transmits ordering and values information from the parent strings to the offspring. A portion of one parent string is mapped onto a portion of the other parent string and the remaining information is exchanged. (A.J. Umbarkar & P.D. Sheth, 2015)

1.3.2.3. Mutation

Like many optimisation algorithms, genetic algorithms carry the risk of being stuck in local optimum solutions, that's where mutation operators come in. The purpose is to change the genomes of offspring generated with crossover to increase the diversity. Increased diversity enables a possibility for the algorithm to jump out of suboptimal solutions and premature convergence. (Lim et al., 2017)

Mutation operators select a genome and change it probabilistically. Although several studies were conducted over the selection of the mutation operators, rate and extent of the mutation is usually problem dependent. (Hong et al., 2002)

An example of a simple mutation operator was used for this application. Known as the 'Random Point Mutation', this operator selects a gene in genome sequence of offspring and replaces it with a random value.

1.4. Previous work

Literature review on procedural generation reveals a large number of research focused on different aspects of gaming as these methods can be applied across different games and genres. For the intentions of this thesis Terrain, Quest, Level, NPC, and Texture/asset generation were reviewed.

1.4.1. Terrain generation

Terrain generation focuses on creating the "physical plane" in which the game takes place. The most notable game that utilizes terrain generation is "Minecraft" where all the terrain blocks and biomes are generated procedurally. A variety of previous research is available for terrain generation.

Himite, utilized randomly generated points to generate Voronoi tessellations to create regions and used noise maps to give context to these regions. He also used Perlin noise to populate the said regions with foliage, rivers, and seas. His work was a major point of inspiration for this thesis. (2022)

Togelius, Preuss and Yannakakis proposed a search-based multi-objective evolutionary algorithm that generates RTS game maps by balancing competing objectives such as asymmetry, resource locations and clustering that were defined by expected player experience. Their algorithm can visualize trade-offs between the objectives, providing a useful tool to game designers. (2010)

Rose & Bakaoukas proposed a method for applying genetic algorithms to create three-dimensional terrain. Their algorithm allows a user to specify rough region boundaries, this is used to create a silhouette which is later accompanied by height generated by a secondary algorithm. They state that splitting the problem into two allows them to modulate the amount of computational power needed. (2016)

1.4.2. Quest generation

Quest generation is one of the earliest fields of applications of procedural generation to cater for RPGs and MMORPGs. As quests are the drivers in stories in many games, different approaches highlighting characters, motivations and stories were proposed.

Soares de Lima et al. came up with a method where they combined planning with an evolutionary search strategy guided by story arcs, their method can generate coherent quests based on individual narrative structures. Their method generates a plot and depending on the steps "tension" is calculated, which is later used for evaluation of their story arc. (2019)

Another noteworthy approach was using quest generation as a context generation of game-space. In this research, authors proposed a lock-key puzzle quest system for its flexibility in a procedural generation context, they are used as a bond between narrative and the game space. The game space was created as connected tiles, as a player reaches the end of one tile another tile is generated with Java backend that is using constraints of player, game, and map state altered by the keys player has. (Ashmore & Nitsche, 2007)

In a direct implementation, Machado has created a system that uses extended grammar to generate quests. The grammar generated defines the motivations, events, NPC's and items. Machado managed to adapt his system into Unreal Engine and embedded the quest generator into quest giver NPC's. Motivations of this NPC guided the motivation of the quest, making it more coherent with the game. (Machado, n.d.)

1.4.3. Level generation

Level generation -sometimes referred to as Dungeon generation- is mostly applied to "Rogue like" games, which are mostly two-dimensional crawlers where usually a player traverses these levels in a quick manner while trying to stay alive.

The generated "level" refers to the creation of the finite game space, that needs to satisfy certain restriction. (i.e., level needs to be traversable, appropriately difficult.)

In a research, Linden surveyed the potential methods and technologies that could be utilised for procedural dungeon generation. In the research they state that, while there are many methods that can be used, there is no specific method that produces better results compared to the other ones.

Linden highlights three categories of procedural generation methods. (van der Linden et al., 2014)

- Genetic Algorithms, the main approach taken in this thesis.

- Cellular Automata, self-organizing structure of a grid of cells in any finite number of dimensions.
- Generative Grammar, a method of creating phrases through finite selection from a list of recursive transformational (or production) rules, which include words as terminal symbols.

Of course, these are not the only methods that are viable as Linden mentions the reliance of methods to the game mechanics and theme, they are producing content for.

In this spirit, Compton and Mateas proposed a system for generating levels specifically for platformer games. They proposed a new four-layer hierarchy to represent platform game levels, with a focus on representing repetition, rhythm, and connectivity which later could be used for generating new levels. (Compton & Mateas, 2006)

Mawhorter and Mateas, later proposed that constraint-based approaches to level design often limited the models resulting in level designs that are not aesthetically pleasing. They called their method Occupancy Regulated Extension relies on pre-authored level chunks as the building material, which later turned into levels with referral of geometry and playability. This method was able to generate more "creative" levels compared to other methods. (Mawhorter & Mateas, 2010)

1.4.4. NPC generation

The applications in the Non-Player Character domain within games are not as numerous as in other fields. Existing applications focus on the behavioural dimension of the NPCs as the challenge here is to not only create coherent characters but have them interact with the player in a consistent way.

Bulitko, Walters, and Brown proposed to model actual players and create AI- controlled proxies to evolve NPCs against. They state that this method not only result in more immersive NPCs who interact and react to player, but also more forgiving in terms of performance as the evolution happens offline. Their current application contains deep neural nets for NPC decisions depending on their observance of the game-space. The simulated evolution does not have discrete generations. Instead, the NPCs reproduce if they are sufficiently old and healthy. (2018)

1.4.5. Texture & asset generation

Texture and assets cover a wide spectrum of applications in games. These are also the applications where that tend to be more specific to the game in question or those that can't be easily categorized.

An example is where, Washburn and Khosmood created a system that generates unique procedural thematic music for non-player characters (NPC) based on developer-defined attributes and game state. The system responds in real-time to the dynamic relationship between the player and target "boss" NPC. This system uses four distinct values arousal, valence, time, and seed. These values together resolve to various aspects of the final musical composition. (2020)

1.5. Thesis objective

Previous research on procedural generation, especially with genetic algorithms were mainly focused on specific games or applications and as such produced tailored results. The narrow approach increases applicability as the issues tackled are parts of essential pipelines in game development process.

This thesis explores the possibility of mimicking the diversity of nature by designing an algorithm that can present solutions (content) as diversely as possible. The specific aim is to create a system that is capable of creating content-blueprints for different aspects of games by having sub-systems that can populate themselves with content (characters, environment, danger, objectives etc.) that is in-line and coherent with each other.

2. PROPOSED SYSTEM

To be able to generate content in a coherent and a viable way, four sub-systems were created. Each of these "engines" are focused on a different aspect of RPGs. In general, these engines create the environment and randomly initialises different populations of individuals for quests, encounters and characters.

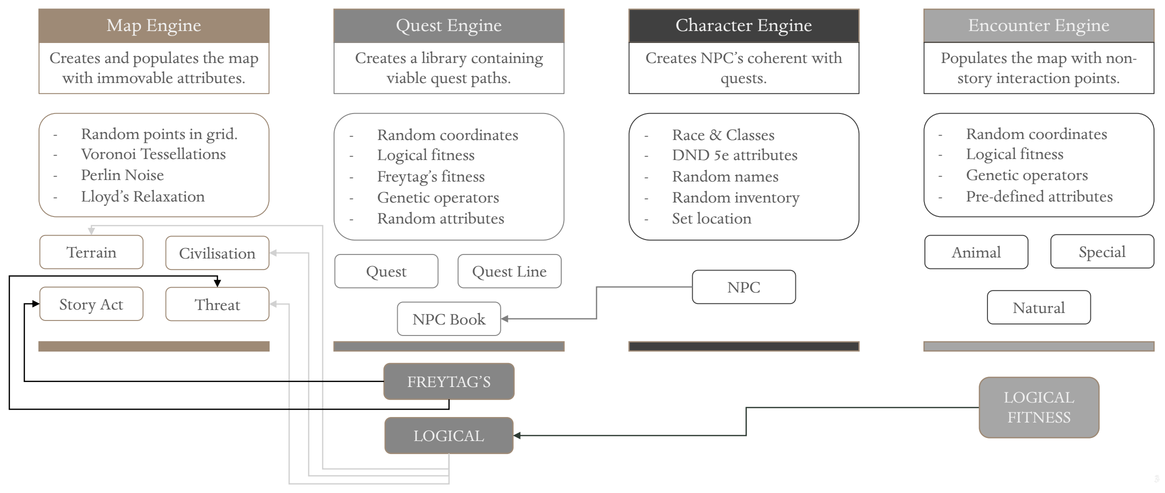


Figure 1: Proposed system overview with sub-class interactions.

Engines use each other as input and output locations, especially for quests and encounters where genetic algorithms were implemented. This connection of systems provide coherency between different views to the map, which normally would require game, level, or quest designers' time. The output for the system is different views on a map generated and populated procedurally. In RPG terms, this would be the game state on beginning.

Engines:

- **Map Engine:** creates and populates the map with base attributes.
- **Quest Engine:** creates a library containing viable quests and quest lines.
- **Character Engine:** creates random NPCs, tied to the generated quests.
- **Encounter Engine:** randomly but coherently populates the map with pre-defined interactions.

Generated populations in different systems tie to each other via coordinates as exact and relative locations for any individuals are used to determine whether they are fit. To be able to create base for this evaluation, a game space needs to be created.

2.1. MAP GENERATION

The game-space or the map, is essentially a grid of points. Each point or location in the map needs to have different views providing information on different game aspects. The generation process has two phases, initialising the regions and assigning attributes to them.

2.1.1. Creating Regions

To generate new random maps whenever needed, the basis of the map needs to be random numbers. For the purposes of this thesis a 1024 x 1024 grid was initialised with 512 random points or “locations”. These points will serve as the centre of a region.

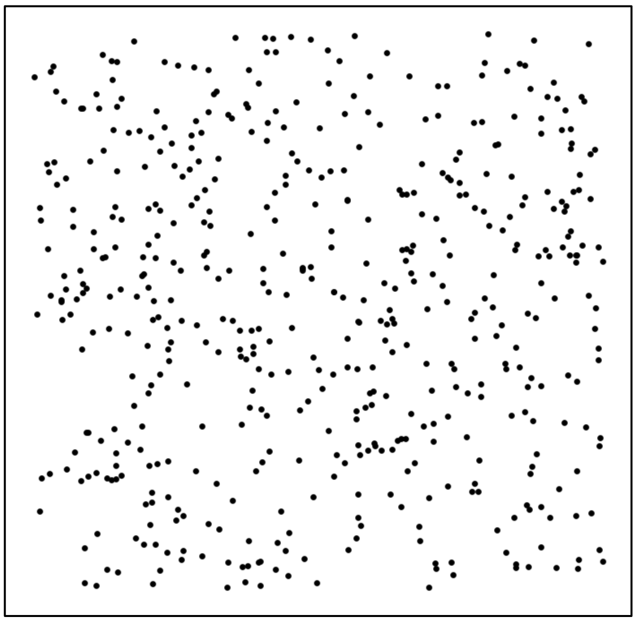


Figure 2: 1024x1024 map grid initialised with 512 randomly generated points.

Each centroid is tied to a region and the area of these regions are calculated as consists of any points closer to that centroid, the area is also called Voronoi Tessellation as for every centroid α ∈ Ω, Ω being the map space, there is a certain point in x ∈ Ω in the map that is closer to α than any other centroid. (Lucarini, 2009)

For this implementation, Voronoi class in scipy.spatial library was utilised as the class is able to store region, ridges and vertices for different tessellation calculations.

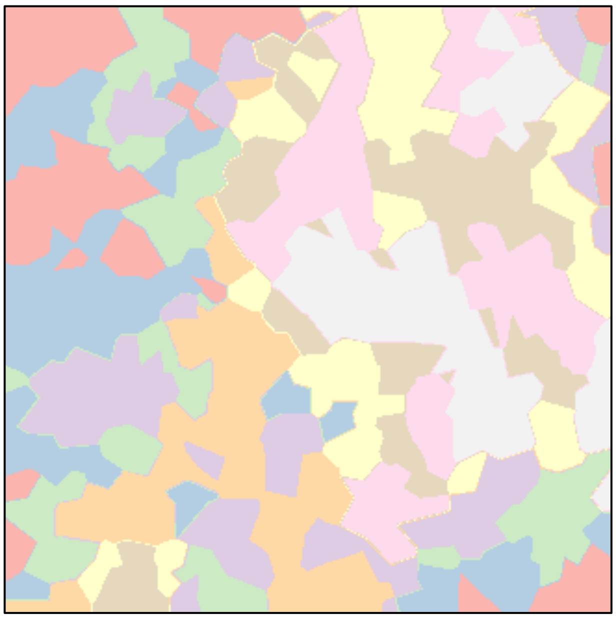


Figure 3: Voronoi tessellations for randomly initialised map, arbitrarily coloured.

Due to nature of initialisation, some clustering between centroids is visible on the map and it doesn't look natural. These points ideally would be uniformly distributed to the map. For this issue, Llyod's Relaxation algorithm was utilised. This iterative algorithm takes the initial locations, moves them to the centre of their cell and re-calculates the Voronoi region resulting in more uniform cells every iteration.

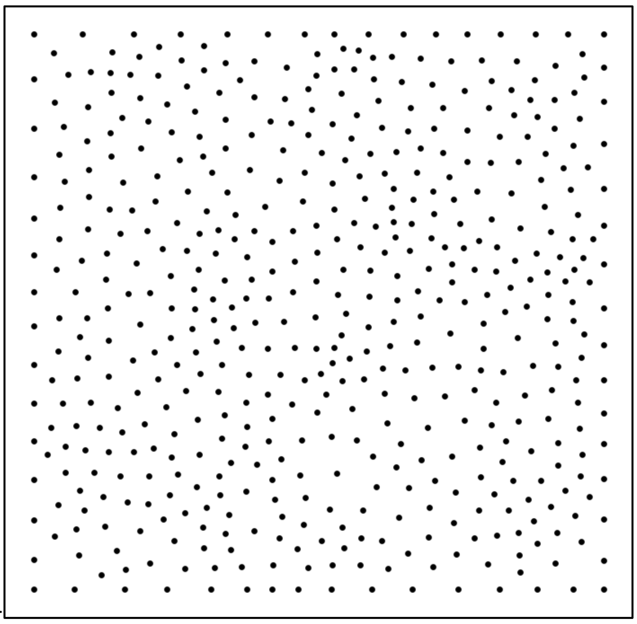


Figure 4: Location points after Llyod's Relaxation algorithm is applied.

To achieve more natural look, the ridges between regions were distorted, the affect is particularly necessary for the terrain view of the map as one does not expect to see straight line on a realistic map.

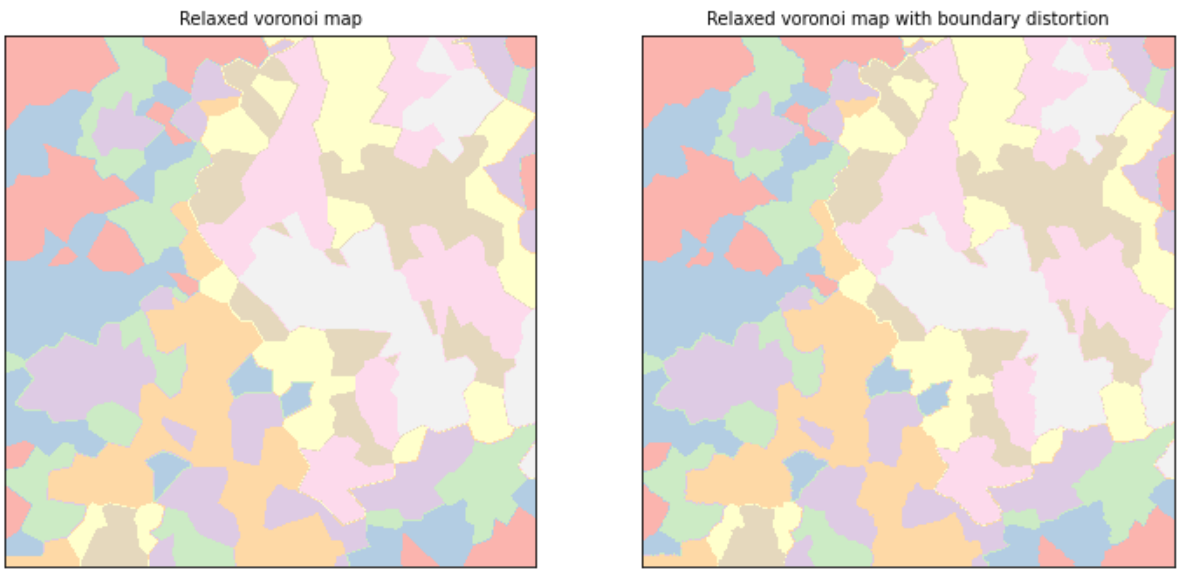


Figure: 5 Voronoi tessellations drawn from relaxed location points before and after boundary distortion, arbitrarily coloured.

2.1.2. Attributing Regions

Upon generation of the regions, providing attributes to these regions are the next step as these attributes will provide context for any game elements residing on that specific region. This operation also needs to stem from randomness to ensure different maps can be generated. Initial approach of generating uniform random numbers will not work as this randomness needs to follow a structure. To solve this issue, Perlin noise was used.

Created by Ken Perlin and won him an Academy Award in the process, Perlin Noise is an implementation of gradient noise in multiple dimensions. The algorithm can produce smooth randomness in multiple dimensions. (Perlin, 1985) Perlin noise determines noise at a point in space by computing a pseudo-random gradient at each of the eight nearest vertices then doing an interpolation. (Lagae et al., 2010) For this implementation, the noise package in Python were utilised. The noise class in the package allows users to tune parameters for the noise maps generated, which has proved useful for the initial development process.

Scale: determines at what distance to view, set as 32.

Octaves: levels of detail for the noise map to have, set as 10.

Lacunarity: adjusts the frequency to affect how much detail is added at each octave, set as 0.8.

Persistence: adjusts the frequency to determine how much each octave contributes to the overall shape, set as 2.

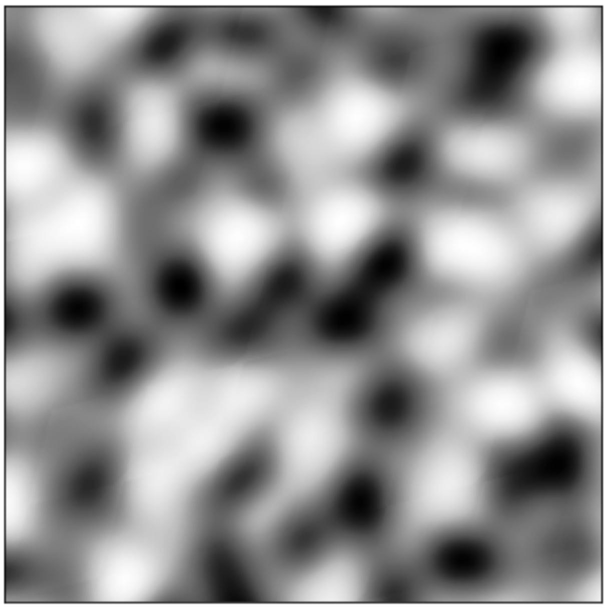


Figure 6: 2-Dimensional Perlin Noise.

As each point in the noise map has a value between -1 to 1, the relative differences of "views" can be determined. These views are pre-defined to generate the levels between them and eventually be used to label the region.

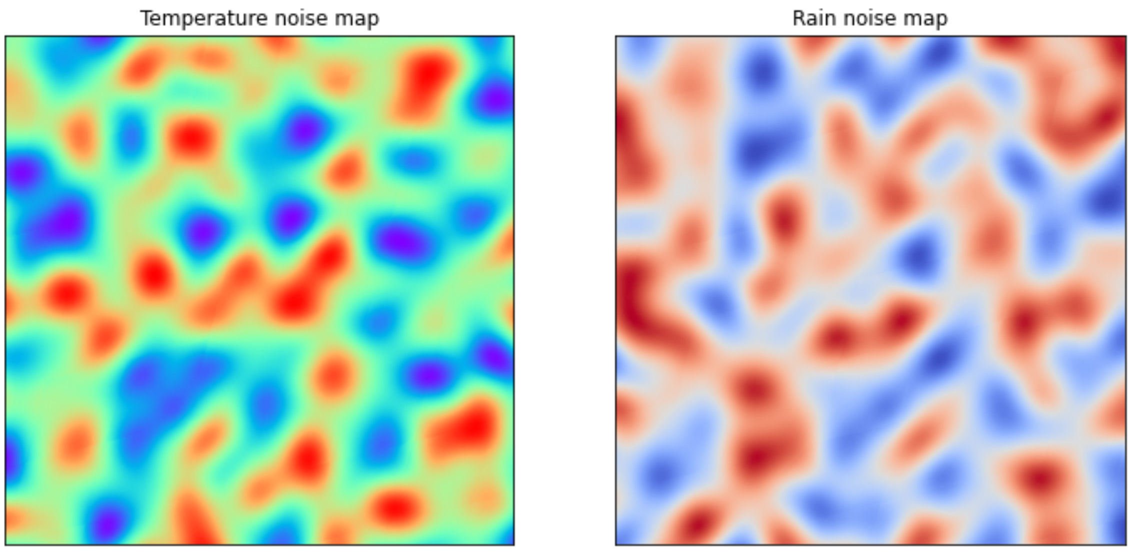


Figure 7: Two noise maps on the same grid, depicting weather related attributes.

Because different points in the regions have different noise levels, to be able to determine the attributes on a regional level, the average noise is calculated for all regions. For each view, the average noise is split into buckets, determining the label. While some views like "Danger" have one noise map determining it, more complex views such as weather uses two noise maps, one for raindrop and one for temperature.

MAP VIEW		Rain noise	Heat noise	Population noise	Threat noise	Act noise
Threat	Threat				1-10	
Terrain	Tundra	0	0			
	Rainforest	2	2			
	Desert	0	2			
	Grassland	1-0	1-2			
	Mountain	2-0	0			
	Forest	2	1			
Civilisation	Wild			1-2-3-4-5-6		
	Countryside			7-8-9		
	City			10		
Story act	Act 0					1
	Act 1					2
	Act 2					3
NOISE LEVEL BUCKETS		**3**	**3**	**10**	**10**	**3**

Table 1: Noise level breakdown for each view and noise map generated for map creation.

With all the different views outlined in Table 1, each point in the map now have context in several dimensions depending on what bucket (low to high) the region in on that specific view. This already creates a coherent map structure where danger level, civilisation level, biome type and story-acts of each location is known.

The number of regions is set differently for each view depending on the context. Lower number of regions results in consolidated areas within map like the Story Act view, regions that needs to have imbalanced distributions are achieved with high number of regions in initialisation. This way, places with highest rate of population covers only a small fraction of the map while story arcs are separated thus avoiding an issue where a player on Act 0 can be stuck between higher Act regions.

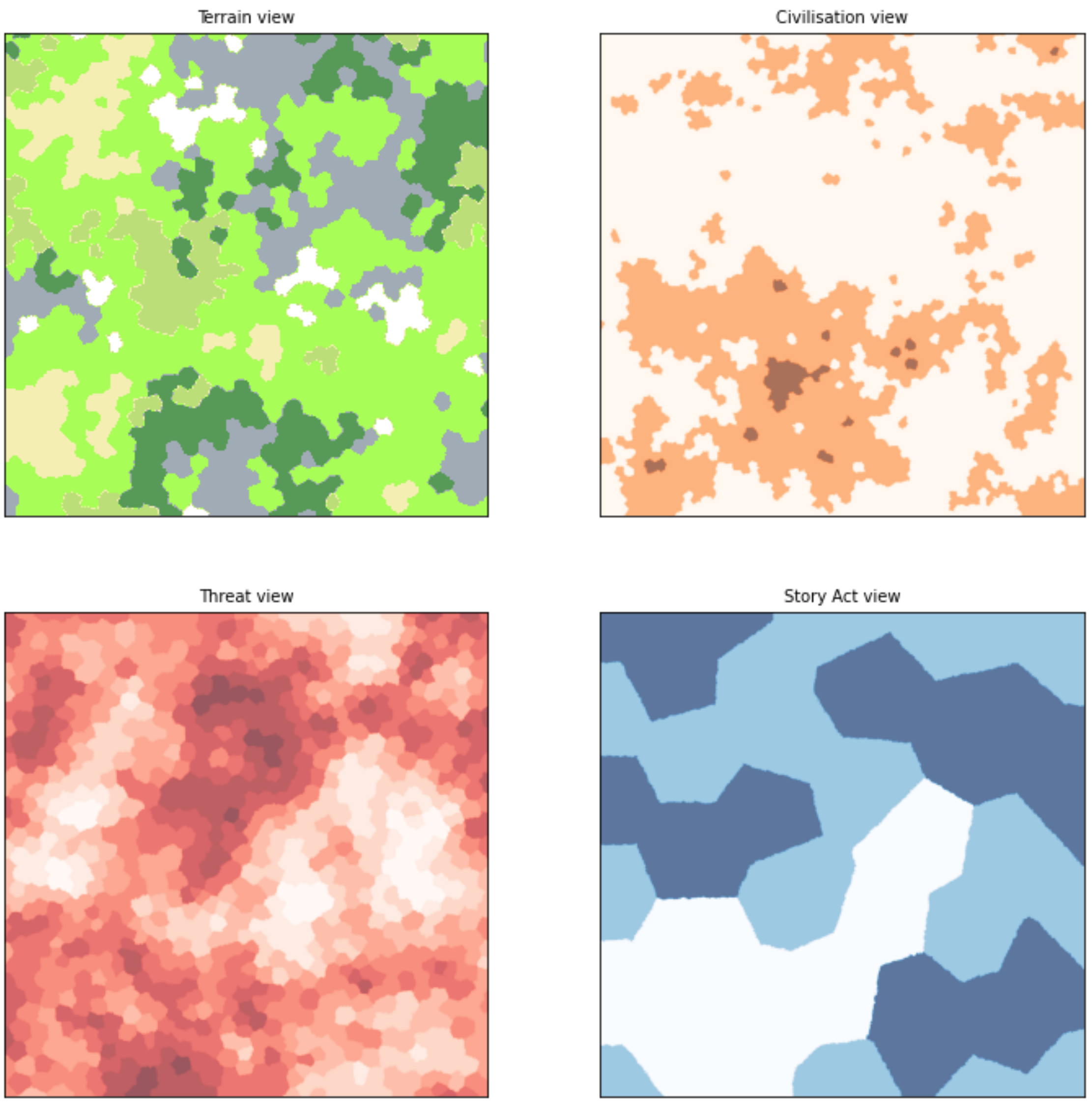


Figure 8: Attributed map regions on different views.

Perlin noise can be further utilised to generate seas within the map with a mask, as any noise value is between -1 and 1, any threshold can be set for the "sea level" giving the user control on the kind of map to generate. Several applications also use the same logic to generate elevation and smaller bodies of water. This application only utilises sea view.

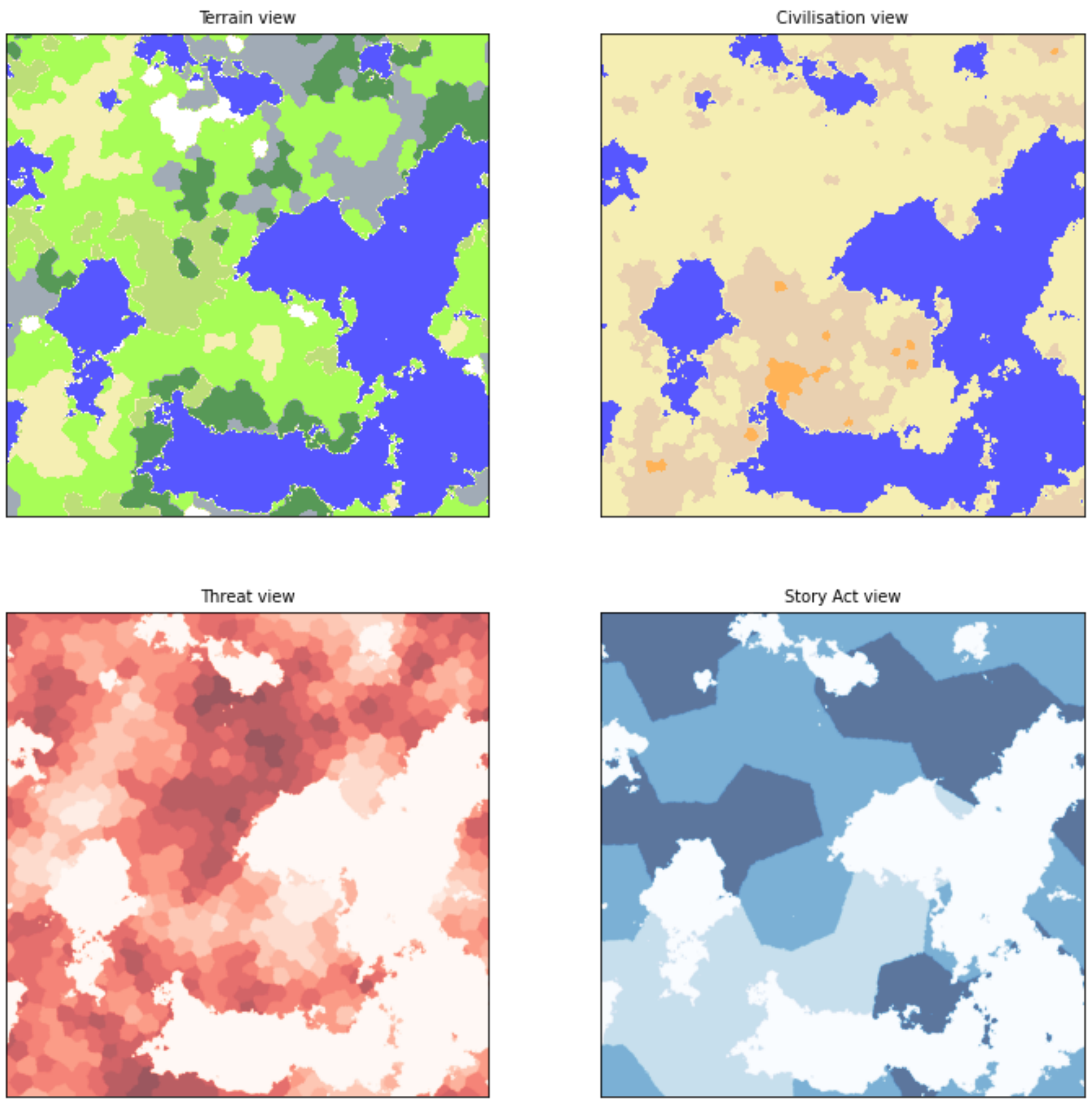


Figure 9: Attributed map regions on different views with sea masks.

With map generation is complete there is a map to populate and attributes to populate it by.

2.2. Encounter Generation

One of the earlier RPG concepts, the encounters cover a wide spectrum of game events that can possibly happen to the player. In earlier games, this mechanic would be triggered with a dice roll. Modern RPGs improved the mechanic to have roaming NPCs around the map that can trigger with an actual encounter in the game space.

Encounters are pre-set events with controlled conditions, one example is getting "ambush" encounter while on a dangerous place. This is done to increase the immersion of the player to the game space.

Encounters	TERRAIN					
	Tundra	Rainforest	Desert	Grassland	Mountain	Forest
tiger	0	1	0	0	0	0
overgrowth	0	1	0	0	0	0
bison	0	0	0	1	0	0
wind_gust	1	0	0	0	0	0
quicksand	0	0	1	0	0	0
donkey	0	0	0	0	1	0

Table 2: Sample of encounter biomes.

Four main types of encounters were defined, and each sub-type of encounters were labelled with a biome in a logical way. (i.e., camel encounter in desert). Differentiation of Encounters are later used on the fitness function calculation specifically using Threat and Civilisation views.

2.2.1. Genetic Algorithm

Encounters needs to be distributed throughout the map however a simple constraint-based algorithm will not be sufficient as each encounter needs to be on the correct location and be naturally distributed to increase immersion.

The proposed genetic algorithm utilises *wencounter* objects. Each *wencounter,* fitness functions are calculated on *wencounter* level but multiple *wencounter* objects or *packs* are treated as individuals. *pack_population* evolves until desired fitness is reached. Only one individual is selected from each *pack* a limited number of fit individuals are taken from each iteration to promote diversity and store them in permanent locations with all other information generated. The fitness function is calculated as follows:

$$F_{encounter}(fitness) = B_{coord} + C_{coord} + T_{coord} + S_{coord} + d_{centroid}$$

$$B_{coord} = \begin{cases} -1000 & if \text{ the coordinate biome is not equal to encounter biome} \\ 0 & \text{otherwise} \end{cases}$$

$$C_{coord} = \begin{cases} -1000 & if \text{ the coordinate in city} \\ -500 & if \text{ coordinate in countryside} \\ 0 & \text{otherwise} \end{cases}$$

$$T_{coord} = \begin{cases} -1000 & if \text{ the encounter is dangerous and coordinate threat is below 3} \\ -1000 & if \text{ the encounter is with safe animals but the threat is above 7} \\ 0 & \text{otherwise} \end{cases}$$

$$S_{coord} = \begin{cases} -1000 & if \text{ the encounter coordinate is in the sea} \\ 0 & \text{otherwise} \end{cases}$$

$$d_{centroid} = \text{Absolute distance to the nearest centroid}$$

Figure 10: The fitness function for encounter generation.

For each *wencounter* the fitness value is calculated by taking correct biome, distance from the centroid of current region, civilisation level and thread level.

Pseudocode for evolution steps:

```
• Initialise the world encounter manager with map input and desired encounter to be generated.
• Initialise a pack population.
• For each type of encounter:
    o While the number of desired encounters (World Atlas length) is not reached.
        ▪ Rank Selection
            • Rank the individual packs depending on their fitness.
            • Assign selection probabilities to each individual, depending on their rank.
            • Select a pair of individuals.
            • Record the selected individuals.
        ▪ Partially Mapped Crossover
            • Create two cut points in the coordinate arrays of parent individuals separately for x and y axes.
            • Replace the coordinate values between the cut points across parents as offspring.
        ▪ Complete Mutation
            • Create random number.
            • If the random number < mutation probability:
                o For all coordinates in the pack generate new random numbers.
            • Else:
                o Pass
        ▪ Check fitness of individual.
        ▪ If desired fitness, append to the World Atlas.
```

The fitness values aside from the distance from centroid centre are penalties multipliers of -500, although several selection operators as Fitness Proportionate Selection and Tournament Selection, because of ease of implementation with negative fitness values, Rank selection was used to select parents.

Among tested Crossover operators, the fastest to converge on optimum solutions were the different versions of Partially Mapped Crossover. Initial design was to have the genome transference on *wencounter* level, but transference proved problematic as it immensely increased variation and prevented the algorithm from converging. To alleviate the variation missing from the crossover for utilising groups of coordinates is compensated with the mutation operators with either tuned to be robust or just with high mutation probability.

The algorithm is run for each combination of encounter types. Through generations and populations fit individuals are recorded to the world atlas, until the desired number of individuals are placed.

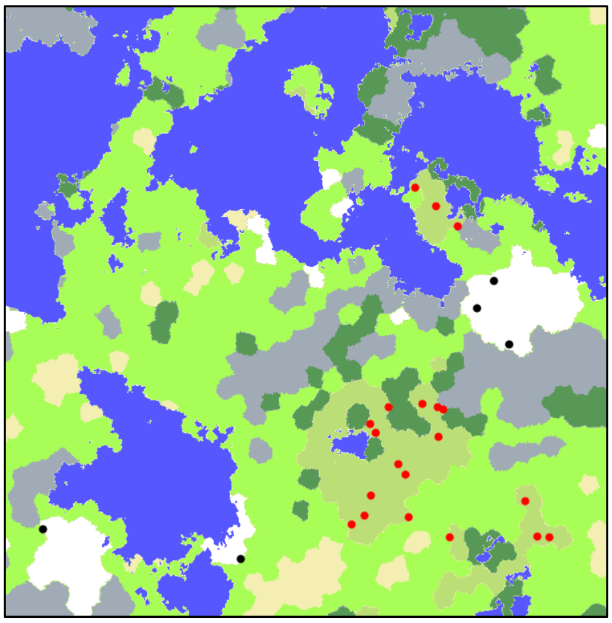


Figure 11: Encounter placement for “Cannibals” and “Snakes”.

Figure X depicts the placement of two sub-groups from Special and Hostile Animal encounters. Black dots depict the “Cannibals” encounter which can happen in dangerous and unpopulated Tundra regions, while red dots depict snakes which can be found in and unpopulated, warm, humid, and dangerous locations.

2.3. Quest Generation

Quests are set of objectives that are given to the player by an interaction within games. Quests are the main point of interaction with story for the player and a big part of the experience. Each quest is a story that player can play and, in some cases, affect the outcome of. In literature there are many different archetypes for stories that can be implemented to the quests.

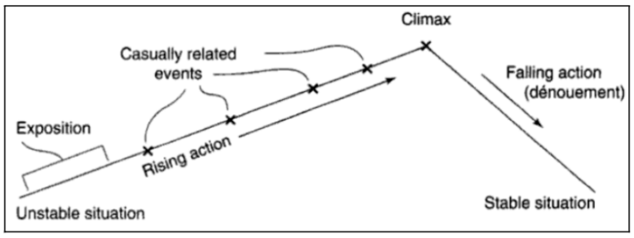
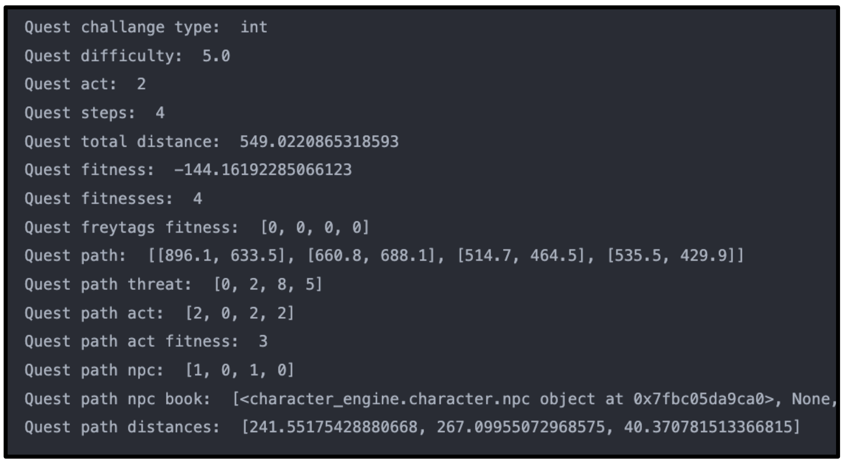


Figure 12: Freytag’s Pyramid

In 1893, renowned German novelist Gustav Freytag developed a narrative structure framework. According to Freytag, dramatic elements are the building blocks in any story and by using this element one should follow the “Five-Act” structure for a good story. These acts are exposition, rising action, climax and falling action. (Rolfe et al., 2010)

In gaming sense, exposition would be the player encountering something out of normal and being introduced to the quest, usually with a quest giver NPC, rising action would be following the leads and encountering more and more dangerous/challenging situations that lead up to the main challenge or climax of the quest, the boss. After the challenge is overcome, a dangerous path to traverse back would be the falling action.

Freytag's Pyramid is already implemented in some studies of quest generation as the basic premise of story elements following a certain structure can be implemented in the genetic algorithms. (Soares De Lima et al., 2019) To be able to evolve and improve, a quest structure needs to be created. Quest object in this application contains the information detailing different aspects of the said quest.



```
Quest challange type:  int
Quest difficulty:  5.0
Quest act:  2
Quest steps:  4
Quest total distance:  549.0220865318593
Quest fitness:  -144.16192285066123
Quest fitnesses:  4
Quest freytags fitness:  [0, 0, 0, 0]
Quest path:  [[896.1, 633.5], [660.8, 688.1], [514.7, 464.5], [535.5, 429.9]]
Quest path threat:  [0, 2, 8, 5]
Quest path act:  [2, 0, 2, 2]
Quest path act fitness:  3
Quest path npc:  [1, 0, 1, 0]
Quest path npc book:  [<character_engine.character.npc object at 0x7fbc05da9ca0>, None,
Quest path distances:  [241.55175428880668, 267.09955072968575, 40.370781513366815]
```

Figure 13: Example quest object attributes.

Figure X shows an example of a random quest object. When a quest object initialises, following attributes are assigned and later used in the GA:

Quest path: Quest path is a set of coordinates that needs to be traversed by the player. It is used to determine the fitness of the quest by calculating distances, checking biomes and regions with different map views.

Challenge type: defines two types of challenges this specific quest has. Challenge types are same as character attributes. If a quest has a challenge type of Strength and Charisma this could mean that it can be resolved diplomatically or with brute force. Challenge type is also used later on to introduce quest NPCs.

Difficulty: Difficulty is calculated over threat view on the map. Especially the locations where the rising action and climax happens sets the difficulty.

Game act: Most games are divided into acts. These acts restrict player from having access to everything all at once and helps them traverse the story better. Game act is decided depending on the location of the quest steps.

Steps: Steps define how many locations needs to be accessed for the quest to be completed.

Distance: The total Euclidian distance between quest steps.

NPC Book: Contains the information of whether an NPC is present in each quest step. Quests are more likely to start with an NPC, the Quest Giver and more likely to have an NPC on the climax, the Boss.

For determining if a quest is good, we have two sets of criteria: Freytag's story structure and quest viability which is used in the GA. This way, any quest is narratively enticing and is possible to play. An example of an undesirable quest would be meeting the boss at the first step and being sent to the middle of the sea to resolve the issue. (Seas are untraversable for this application.) However, having singular quests alone is not enough as games tells stories through interconnected quests spanning across game, this is called a quest line.

An example of a two-step quest line would be in the first quest, player would be stealing back an item from bandits for an NPC. The second quest shows up later in the game when the original NPC is attacked for revenge by the same bandit group. The end goal here is to create coherent quests that follow each other through game progression. For performance reasons this goal is split into two algorithms.

2.3.1. Quest Library Generation

The first step is to create the singular viable quests that can be later compiled into a library to be curated from later. This way, any quest line can be generated on demand. The fitness function for the quest generation algorithm uses penalisation.

For assessing Freytag's fitness, each combination is varying length is taken and each step in the quest is compared to each other. This helps determine how many steps is out of place in terms of danger. Like the logic followed with *wencounters* and *packs*, this allows the algorithm to converge faster. For the viability of quests several different logical penalties were applied. Evolving process for different steps and acts of quests are done separately to ensure there is enough quests with different properties in the quest library and to reduce the computation time. The fitness function for the first step is as below.

$$F_{quest}(fitness) = Cd_{coord} + Act_{com} + Act_{max} + Fr_{fitness} + S_{coord} + d_{civilisation} + d_{avg}$$

$$Cd_{coord} = \begin{cases} -1000 & \textit{if} \text{ the begining coordinate is in a region with low civilisation score} \\ 0 & \text{otherwise} \end{cases}$$

$$Act_{com} = \begin{cases} -1000 & \textit{if} \text{ the most common story act places in quest is not equal to intended quest act} \\ 0 & \text{otherwise} \end{cases}$$

$$Act_{max} = \begin{cases} -1000 & \textit{if} \text{ the maximum story act present in quest is higher than the intended quest act} \\ 0 & \text{otherwise} \end{cases}$$

$$Fr_{fitness} = \begin{cases} -1000 & \textit{for each} \text{ quest location that does not follow the Freytag's Pyramid in terms of danger} \\ 0 & \text{otherwise} \end{cases}$$

$$S_{coord} = \begin{cases} -1000 & \textit{if} \text{ the encounter coordinate is in the sea} \\ 0 & \text{otherwise} \end{cases}$$

$$d_{avg} = \begin{cases} -1000 & \textit{if} \text{ the average distance between quest steps is } \geq 200 \\ 0 & \text{otherwise} \end{cases}$$

$$d_{civilisation} = \text{Absolute distance to the nearest region with civilisation level} \geq 7$$

Figure 14: The fitness function for the quest generation.

Selection is done similarly to the Encounter Engine. However especially in the earlier generations, it was challenging to have proper individuals selected as the fitness was the same for a lot quests. To overcome this issue, individuals are first ranked by their actual fitness values then they are also ranked by fitness of the story act, fitness of suitability and total distance. This way aspects that the algorithm was struggling with was given an incentive to be selected.

Pseudocode for evolution steps:

```
• While the number of desired quests is not reached.
    o Initialise the quest population object.
    o Start evolving process.
        ▪ Rank Selection
            • Rank the individual packs depending on their fitness.
            • Assign selection probabilities to each individual, depending on their
              rank.
            • Select a pair of individuals.
            • Record the selected individuals.
        ▪ Partially Mapped Crossover
            • Create two cut points in the coordinate arrays of parent individuals
              separately for x and y axes.
            • Replace the coordinate values between the cut points across parents
              as offspring.
        ▪ Random Point Mutation
            • Create random number.
            • If the random number < mutation probability:
                o Select a random step in quest and replace the coordinates
                  with a random value.
            • Else:
                o Pass
        ▪ Check fitness of individuals.
        ▪ If desired fitness:
        ▪ Break
```

This results in a library filled with individual quests that are “fit” with varying lengths, danger, and locations.

2.3.2. Quest Line Generation

To be able to create flowing quest lines, a separate less complex Genetic Algorithm was created. This second algorithm only takes Freytag’s fitness and the max act (Quest step with the highest story act.) is taken. Logically all quest lines would start at a certain act and continue across different acts. Enabling a time or a barrier between quests. Curation function arguments story act and number of quests required in the quest line. As the first algorithm ensured the viability of all quests, curation algorithm utilises relatively simple operators. Fitness is decided by only the sequence of these quests and does not need mutation to converge on a viable quest line.

The fitness function was defined as follows where:

$$F_{quest\_line}(fitness) = Act_{max} + Act_{min} + Fr_{fitness}$$

$$Act_{max} = \begin{cases} -1000 & \textit{if} \text{ the max story act present in quest is not equal to specified maximum act} \\ 0 & \text{otherwise} \end{cases}$$

$$Act_{min} = \begin{cases} -1000 & \textit{if} \text{ the min story act present in quest is not equal to specified minimum act} \\ 0 & \text{otherwise} \end{cases}$$

$$Fr_{fitness} = \begin{cases} -1000 & \textit{for each} \text{ quest that does not follow the Freytag's Pyramid in terms of difficulty} \\ 0 & \text{otherwise} \end{cases}$$

Figure 15: Fitness function for quest line generation.

Pseudocode for evolution steps:

```
• Initialise the quest library object with GA parameters.
    o Run the quest generator to populate the library with viable quests.
• For all story acts:
    o For all quest steps:
        ▪ While no viable quest line is found:
            • Rank Selection:
                o Rank the individual quest lines depending on their fitness.
                o Assign selection probabilities to each individual, depending on
                  their rank.
                o Select a pair of individuals.
                o Record the selected individuals.
            • Single Point Crossover:
                o Create a cut point within the quest line.
                o Exchange the individual quests with each other depending on
                  the cut point.
            • Check fitness of individuals.
            • If desired fitness:
                o Break
```

2.4. Character Generation

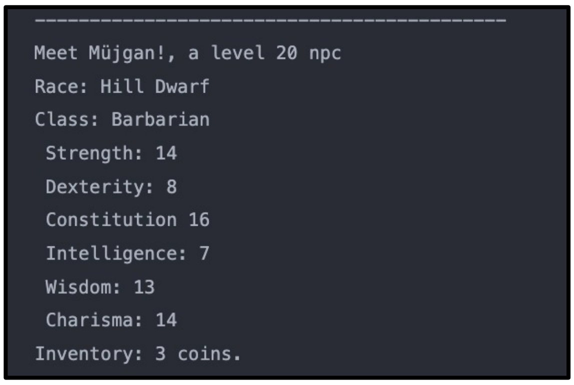
Dungeons and Dragons is a tabletop fantasy role-playing game. Most players role-play adventuring characters such as an elves or wizards. The game is facilitated completely on imagination and to help players have a structure for their imagination it includes many systems such as Health Points, Combat, Character Attributes and Economy. (Gygax et al., 1978)

DnD has been a major point of inspiration for this thesis, especially on the implementation of the character engine. In DnD, characters are created with both imagination and systemic constraints such as level, race, and class. A low-level character won't be as strong as a higher level one and a character that is extremely smart, will have disadvantages on other attributes. Character creation starts with picking a race and a class, these will determine the aptitude of the character to certain traits (i.e., Dwarves are physically strong) and set attributes as size, speed, spoken languages.

In DnD interactions are resolved depending on character attributes. On any given situation, players roll a dice and depending on the situation they will have advantage or disadvantage. If they manage to score higher with their role and ability score modifiers, they will succeed. These attributes are:

- Strength (STR): Determines how physically strong a player is.
- Dexterity (DEX): Measures actions with agility such as acrobatics, sleight of hand or stealth.
- Constitution (CON): Measures the endurance character, important for Health Point system.
- Intelligence (INT): Related to actions that involves knowledge, memory, and magic.
- Wisdom (WIS): The knowledge of life as animal handling, insight, medicine, and perception.
- Charisma (CHR): Aptitude of interaction with characters as lying, intimidating, and persuading.

The DnD system implemented in this thesis assigns a base value of 8 to all attributes then randomly distributes limited number of points across these attributes, this ensures the diversity of characters. Although DnD has a very complex level system, levelling up only affects the base attributes for the character in this implementation. (*Basic D&D Rules | Dungeons & Dragons*, n.d.)



```
--------------------------------------------
Meet Müjgan!, a level 20 npc
Race: Hill Dwarf
Class: Barbarian
 Strength: 14
 Dexterity: 8
 Constitution 16
 Intelligence: 7
 Wisdom: 13
 Charisma: 14
Inventory: 3 coins.
```

Figure 16: Example NPC attributes.

Figure XX depicts a random character created by the Character Engine, because Barbarian Dwarves are prone to be strong and resilient with the bonusses, this character has high Strength and Constitution. Rest of the random distribution results in a character that is not very smart but very charismatic. The DnD system allows for controlled randomness as any characters generated is sensible in confines of the game but are different than each other. This is one of the main drivers of the character motivation and storytelling.

NPC creation the Character Engine is driven by the Quest Engine, each viable quests generated contains a Boolean array called the NPC Book, determining whether an NPC is present on that location. The challenge type and NPC Book determines the character generation. This process happens when the quest library is generated to avoid unnecessary computation during quest gsssseneration.

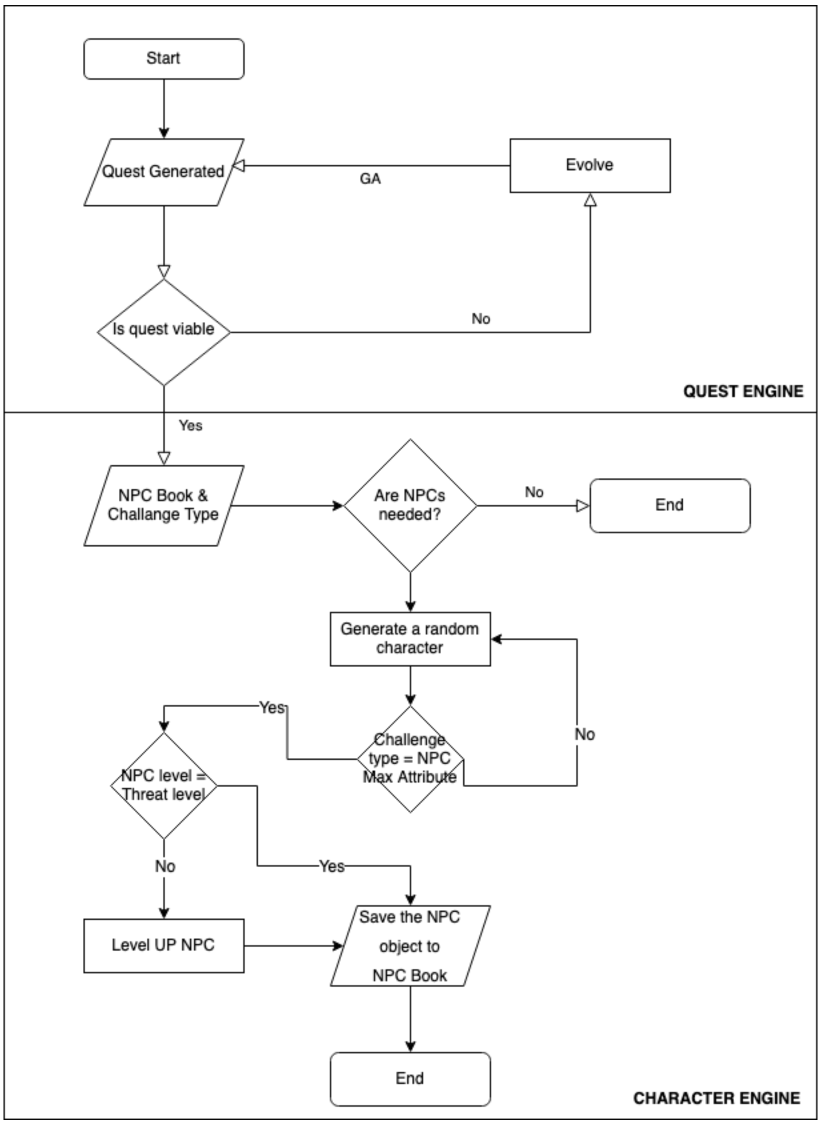


Figure 17: Quest NPC generation algorithm flow.

When a quest placed in the Quest Library, Character Engine is called to create a random character whose highest attribute matches the challenge type of the quest. Once this character is generated, it is levelled up. Characters can gain attribute bonusses from +2 to +6 depending on the level bracket they are in. This also ensures variability in the sense that placed character will be proficient in the challenge type attribute but can have other attributes boosted with danger. An example to this is an intelligence challenge in a high danger location could have a wizard character with high intelligence and boosted constitution. In a simple story this would be hinting that the wizard has been living in hostile conditions and got accustomed to it.

The algorithm for character generation is relatively simple yet it provides the main aspect for the output to be used as a blueprint. All engines together ensure coherency in terms of the game and space, this alleviates the burden of world building and tying all elements together for a storyteller or a game designer.

3. RESULTS & CONCLUSIONS

The main objective of this thesis was the evaluate the possibility of generating content with Genetic Algorithms. In traditional optimisation approach, the optimum solution is always sought after. The whole process of evolution is focused on getting desirable levels, in the process countless of sub-optimal solutions with a major diversity is created. The aim of the generated system was to focus on all viable solutions within the search space. The main approach of using penalisation in fitness functions proved to be useful to achieve this result.

As expected, the simple and existing Genetic Operators proved very effective, the simplicity of the operators is desired as they impact the performance of the system substantially and if they can converge on any solution, they produced desirable results. One downside of setting the functions with constraints were having static penalties that could not converge on a solution as all inputs in the algorithm carried the same weight and no penalties could be overlooked, a dynamic way of measuring these fitnesses was only possible through distances. Unviable distances were still penalised but having the distance to centroids, cities and each other as measure between viable individuals helped the algorithm to be able to find the best individuals.

The other side of this coin was Gaming, specifically the embedded story and mechanical aspects on games was the focus of this thesis. Every addition of different gaming aspect to the system have improved it more than the Genetic Operators. Fitness functions generated and every constraint added to the functions increased the dimension of complexity. Especially inputs from different domains such as literature made all the difference. Tying the Quest Generator and Character Generator into the map would not be possible without Freytag's narrative theories. Similar inputs from psychology, political science, macro-economic can also benefit this system immensely because the systems embedded within games are already based on reality, having the algorithm created with the same inspiration not only viable but it also allows for a more general use for the procedural generation systems created. A model capable of creating an outline of game economy will be far more effective than a system that tries to calculate the in-game currency ramp-up in Fallout.

The general approach also removes limitations of gaming as the produced results from the developed system is a blueprint filled with story beats and progression which theoretically can be used to create any kind of story or world building. The blueprint can alleviate some of the talent and time requirements to create a good coherent world within a story as the system creates the world first and then paints out the possible stories that can be told within it.

Same benefit of viable diversity can also be utilised for game testing. Due to the nature of some games, especially AAA rated releases require a massive amount of testing before release. The games that were not tested properly for their size faces backlash from the players often, recent examples to this would be the release of Fallout 76 and Cyberpunk. After these games were released, exploits and bugs that were not identified by the development team were found by the massive player bases, which resulted in PR damage and lost revenues. If designed accordingly, Genetic Algorithms can simulate all the difference ways of achieving a goal in a limited space by giving valuable hints to game developers on how many ways people can interact with their product with a fraction of costs and time.

4. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

The initial plan for this thesis was far more ambitious with creating a playable game, that has all its elements procedurally generated with the added input from the player progressing through the game. Many of these elements were altered or removed from the final system as the amount of time required to study and develop those concepts would overshadow the actual goal of the thesis.

Completion of this work mostly relied on concepts out of data science, by providing the perspective or context they progressed and shaped the system to its current state. It is highly recommended for future works to be conducted with a team from different disciplines, as it is different perspectives that allows for abstract concepts to be mathematically and algorithmically represented. The technical challenges of this type of work can be avoided by building a system on a functional game engine as this was done by several studies in the field.

After the development of the system, a proper way to measure the quality of the content posed a major issue. In existing literature, this was only done for work that was embedded in the games via play-testers. The only viable testing for this application, was through the inner workings of the algorithm.

As there is no optimisation algorithm that can solve all optimisation problems, not all content need can be satisfied with one procedural generation algorithm. The complexity of the algorithm rises exponentially with every other dynamic added to the mix. Given the complexity of modern video games, the amount of time necessary to develop an algorithm to generate viable content is not viable. This application should be further investigated for mobile games, mini-games or mechanics within video games and general story telling.

5. BIBLIOGRAPHY

A.J., U., & P.D., S. (2015). CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. *ICTACT Journal on Soft Computing*, *06*(01), 1083–1092. https://doi.org/10.21917/ijsc.2015.0150

Basic D&D Rules | Dungeons & Dragons. (n.d.). D&D Official | Dungeons & Dragons. https://dnd.wizards.com/what-is-dnd/basic-rules

Calvin Ashmore & Michael Nitsche. (2007). The Quest in a Generated World. *Digital Games Research Association Conference*, *4*. http://homes.lmc.gatech.edu/%7Enitsche/download/AshmoreNitsche_DiGRA_07.pdf

Carter, R. R., & Lester, D. (1998). Personalities of Players of Dungeons and Dragons. *Psychological Reports*, *82*(1), 182. https://doi.org/10.2466/pr0.1998.82.1.182

Deep, K., & Thakur, M. (2007). A new mutation operator for real coded genetic algorithms. *Applied Mathematics and Computation*, *193*(1), 211–230. https://doi.org/10.1016/j.amc.2007.03.046

Ebert, D. S., Musgrave, K. F., Peachey, D., Perlin, K., & Worley, S. (2002). *Texturing and Modeling, Third Edition: A Procedural Approach (The Morgan Kaufmann Series in Computer Graphics)* (3rd ed.). Morgan Kaufmann.

Goldberg, D. E. (1986). The Genetic Algorithm Approach: Why, How, and What Next? *Adaptive and Learning Systems*, 247–253. https://doi.org/10.1007/978-1-4757-1895-9_17

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

Gygax, G., Sutherland, D. C., & Trampier, D. A. (1978). *Advanced Dungeons & Dragons, Players Handbook: Special Reference Work : a Compiled Volume of Information for Players of Advanced Dungeons & Dragons, Including, Character Races, Classes, and Level Abilities; Spell Tables and Descriptions; Equipment Costs; Weapons Data; and Information on Adventuring*. TSR Hobbies.

Himite, B. (2022, January 4). *Replicating Minecraft World Generation in Python - Towards Data Science*. Medium. Retrieved October 22, 2022, from https://towardsdatascience.com/replicating-minecraft-world-generation-in-python-1b491bc9b9a4

Holland, J. H. (1992a). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Amsterdam University Press.

Holland, J. H. (1992b). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Amsterdam University Press.

Hong, T. P., Wang, H. S., Lin, W. Y., & Lee, W. Y. (2002). Evolution of Appropriate Crossover and Mutation Operators in a Genetic Process. *Applied Intelligence*, *16*(1), 7–17. https://doi.org/10.1023/a:1012815625611

Kate Compton & Michael Mateas. (2006). Procedural level design for platform games. *National Conference on Artificial Intelligence*, 109–111. http://aaaipress.org/Papers/AIIDE/2006/AIIDE06-022.pdf

Kelkar, A., Dahibhate, M., & Jagtap, S. (2022). Procedural Foliage Generation. *International Journal for Research in Applied Science and Engineering Technology*, *10*(5), 1628–1632. https://doi.org/10.22214/ijraset.2022.42410

Koetsier, J. (2020, September 26). *Global Online Content Consumption Doubled In 2020*. Forbes. Retrieved October 22, 2022, from https://www.forbes.com/sites/johnkoetsier/2020/09/26/global-online-content-consumption-doubled-in-2020/?sh=71fd527c2fde

Kora, P., & Yadlapalli, P. (2017). Crossover Operators in Genetic Algorithms: A Review. *International Journal of Computer Applications*, *162*(10), 34–36. https://doi.org/10.5120/ijca2017913370

Lagae, A., Lefebvre, S., Cook, R. L., DeRose, T., Drettakis, G., Ebert, D. S., Lewis, J. S., Perlin, K., & Zwicker, M. (2010). A Survey of Procedural Noise Functions. *Computer Graphics Forum*, *29*(8), 2579–2600. https://doi.org/10.1111/j.1467-8659.2010.01827.x

Lambora, A., Gupta, K., & Chopra, K. (2019). Genetic Algorithm- A Literature Review. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. https://doi.org/10.1109/comitcon.2019.8862255

Lim, S., Sultan, A. B., Sulaiman, M. N., Mustapha, A., & Leong, K. Y. (2017). Crossover and Mutation Operators of Genetic Algorithms. *International Journal of Machine Learning and Computing*, *7*(1), 9–12. https://doi.org/10.18178/ijmlc.2017.7.1.611

Lucarini, V. (2009). Symmetry-Break in Voronoi Tessellations. *Symmetry*, *1*(1), 21–54. https://doi.org/10.3390/sym1010021

Machado, A. S. F. (2017). *A procedural quest generator for Conan Exiles* [M.Sc. Thesis]. University of Lisbon.

Mawhorter, P., & Mateas, M. (2010). Procedural level generation using occupancy-regulated extension. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. https://doi.org/10.1109/itw.2010.5593333

Perlin, K. (1985). An image synthesizer. *Computer Graphics*, *19*(3), 287–296. https://doi.org/10.1145/325165.325247

Rolfe, B., Jones, C. M., & Wallace, H. M. (2010). Designing Dramatic Play: Story and Game Structure. *Electronic Workshops in Computing*. https://doi.org/10.14236/ewic/hci2010.54

Rose, T. J., & Bakaoukas, A. G. (2016). Algorithms and Approaches for Procedural Terrain Generation - A Brief Review of Current Techniques. *2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*. https://doi.org/10.1109/vs-games.2016.7590336

Short, T., & Adams, T. (2017). *Procedural Generation in Game Design*. Amsterdam University Press.

Singer, D., & D’Angelo, E. (2021, November 4). The Netflix of gaming? Why subscription video-game services face an uphill battle. *McKinsey & Company*. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-netflix-of-gaming-why-subscription-video-game-services-face-an-uphill-battle

Soares De Lima, E., Feijo, B., & Furtado, A. L. (2019). Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning. *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. https://doi.org/10.1109/sbgames.2019.00028

The evolution of business models in the video-game industry. (n.d.). EDHEC BUSINESS SCHOOL. Retrieved October 22, 2022, from https://www.edhec.edu/en/news/evolution-business-models-video-game-industry

Togelius, J., Preuss, M., & Yannakakis, G. N. (2010). Towards multiobjective procedural map generation. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games - PCGames ’10*. https://doi.org/10.1145/1814256.1814259

Vadim Bulitko, Mac Walters, & Matthew R. G. Brown. (2018). Evolving NPC Behaviours in A-life with Player Proxies. *AIIDE Workshops*. http://ceur-ws.org/Vol-2282/EXAG_116.pdf

Van Der Linden, R., Lopes, R., & Bidarra, R. (2014). Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*(1), 78–89. https://doi.org/10.1109/tciaig.2013.2290371

Washburn, M., & Khosmood, F. (2020). Dynamic Procedural Music Generation from NPC Attributes. *International Conference on the Foundations of Digital Games*. https://doi.org/10.1145/3402942.3409785

Yao, X. (1993). An empirical study of genetic operators in genetic algorithms. *Microprocessing and Microprogramming*, *38*(1–5), 707–714. https://doi.org/10.1016/0165-6074(93)90215

6. APPENDIX

Race	STR	DEX	CON	INT	WIS	CHA	Found	Size	Speed
Dragonborn	2					1	Uncommon	Medium	30
Hill Dwarf			2		1		Common	Medium	25
Mountain Dwarf	2		2				Common	Medium	25
High Elf		2		1			Common	Medium	30
Wood Elf		2			1		Common	Medium	30
Drow Elf		2				1	Uncommon	Medium	30
Forest Gnome		1		2			Uncommon	Small	25
Rock Gnome			1	2			Uncommon	Small	25
Half-Orc	2		1				Uncommon	Medium	30
Lightfoot Halfling		2				1	Common	Medium	30
Stout Halfling		2	1				Common	Medium	30
Human	1	1	1	1	1	1	Common	Medium	30
Tiefling				1		2	Uncommon	Medium	30

Table 3: Character races and atribute modifiers.

Encounters	TERRAIN					
	Tundra	Rainforest	Desert	Grassland	Mountain	Forest
tiger	0	1	0	0	0	0
snow	1	0	0	0	0	0
overgrowth	0	1	0	0	0	0
bison	0	0	0	1	0	0
wind_gust	1	0	0	0	0	0
quicksand	0	0	1	0	0	0
donkey	0	0	0	0	1	0
macaw	0	1	0	0	0	0
meerkat	0	0	1	0	0	0
hunters	0	0	0	1	0	0
lion	0	0	1	0	0	0
ravine	**1**	**0**	**0**	**0**	**1**	0
capybara	0	1	0	0	0	0
scorpion	0	0	1	0	0	0
climate_activists	0	1	0	0	0	0
gazelle	0	0	0	0	1	0
sloth	0	1	0	0	0	0
jaguar	0	1	0	0	0	0
monk	0	0	0	0	1	0
eagle	0	0	1	0	0	0
tourists	0	0	0	0	1	0
squirrel	0	0	0	0	0	1
frozen_figure	1	0	0	0	0	0

lizard	0	0	1	0	0	0
heat_wave	0	0	1	0	0	0
wolf	1	0	0	1	0	0
tuskan_raiders	0	0	1	0	0	0
flood	0	1	0	0	0	0
rabbit	0	0	0	0	0	1
wolverine	0	0	0	0	1	0
poison_dart_frog	0	1	0	0	0	0
bobcat	0	0	1	0	0	0
tortoise	0	0	1	0	0	0
animal_migration	0	0	0	1	0	0
sand_dune	0	0	1	0	0	0
goat	0	0	0	0	1	0
mosquitos	0	1	0	0	0	0
maze	0	1	0	0	0	0
coyote	0	0	1	0	0	0
spiky_canopy	0	1	0	0	0	0
rattlesnake	0	0	1	0	0	0
orangutan	0	0	0	0	0	1
leopard	0	0	0	0	1	0
gecko	0	0	0	1	0	0
avalanche	0	0	0	0	1	0
mountain_lion	0	0	0	0	1	0
horse	0	0	0	1	0	0
cliff	0	0	0	0	1	0
cannibals	1	0	0	0	0	0
wild_boar	0	0	0	0	0	1
cult_meeting	0	0	0	0	0	1
wilds	0	1	0	0	0	0
wood_choppers	0	1	0	0	0	0
arctic_hare	1	0	0	0	0	0
hare	0	0	0	0	1	0
snake	0	1	0	0	0	0
bear	1	0	0	0	1	0
arctic_fox	1	0	0	0	0	0
frozen_lake	1	0	0	0	0	0
elephant	0	0	0	1	0	0
dog	0	0	0	1	0	0
camel	0	0	1	0	0	0
fallen_tree	0	0	0	0	0	1
witches_house	0	0	0	0	0	1
parrot	0	1	0	0	0	0
thunderstorm	0	0	0	0	1	0
tapir	0	1	0	0	0	0
spider_monkey	0	1	0	0	0	0
iguana	0	1	0	0	0	0
sand_storm	0	0	1	0	0	0
gopher	0	0	0	1	0	0
fury_road	0	0	1	0	0	0

deer	0	0	0	1	0	1
fire_ant	0	1	0	0	0	0

Table 4: World encounters and their biomes.

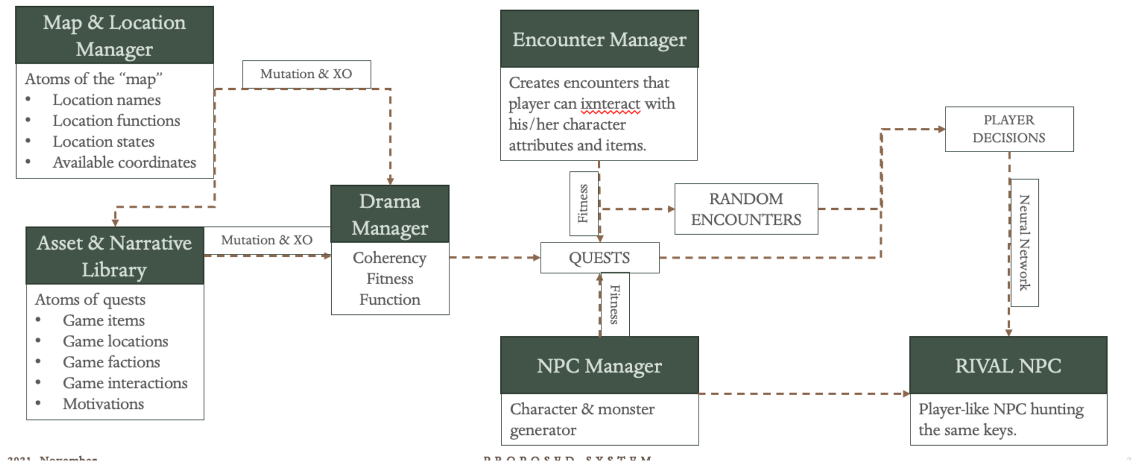


Figure 18: Initial plan for the system.