Simulating human civilisation and the evolution of society based on factors such as culture, religion, and technology

James Benjamin Brimelow Gorman, 100505349

College of Engineering & Technology, University of Derby

100505349@unimail.derby.ac.uk

**Abstract:** LOREM IPSUM

**Keywords:** LOREM IPSUM

Contents

[1 Introduction 3](#_Toc94716206)

[2 Literature Review 3](#_Toc94716207)

[2.1 Dwarf Fortress 3](#_Toc94716208)

[2.1.1 World Generation 4](#_Toc94716209)

[2.2 Europa Universalis 4 4](#_Toc94716210)

[2.2.1 Provinces 4](#_Toc94716211)

[2.2.2 Artificial Intelligence 5](#_Toc94716212)

[2.2.3 Random New World 5](#_Toc94716213)

[2.3 Sid Meier’s Civilization 5 6](#_Toc94716214)

[2.3.1 Artificial Intelligence 6](#_Toc94716215)

[2.4 Iron Age 7](#_Toc94716216)

[2.4.1 World Generation 7](#_Toc94716217)

[2.4.2 Simulation & Artificial Intelligence 8](#_Toc94716218)

[2.4.3 Tile and National Names 8](#_Toc94716219)

[3 World Design 8](#_Toc94716220)

[3.1 World Map 9](#_Toc94716221)

[3.1.1 World Generation 9](#_Toc94716222)

[3.1.2 Provinces 10](#_Toc94716223)

[3.2 World Properties 11](#_Toc94716224)

[4 World Generation 12](#_Toc94716225)

[4.1 Biomes 13](#_Toc94716226)

[5 World Properties 15](#_Toc94716227)

[5.1 Cultures & Names 15](#_Toc94716228)

[5.2 Population 17](#_Toc94716229)

[6 Bibliography 18](#_Toc94716230)

1. Introduction

LOREM IPSUM

1. Literature Review

The uniquity of this project comes in the form of its ability to put existing concepts such as world generation and time simulation together – software already exists to complete some of the objectives of this artefact, but little has been done to collate these ideas into one package. In order to perfect the individual components of the project, analysis into similar projects was completed.

* 1. Dwarf Fortress

The first similar system identified was the 2006 game Dwarf Fortress (Tarn Adams, 2006). This game, which began development in 2002, is set in a world populated by fantasy creatures such as elves, dwarves, and goblins. The objective of the player is to build a new colony for their chosen dwarven civilisation, trade with nearby populations and accumulate wealth. The game is notorious for its difficulty and attention to detail, particularly in its simulation of the world.

The game world is generated by the player at the beginning of the game and uses an entirely random map fit with its own history and civilisations, as well as records such as that of key peoples and artefacts. This, much like the artefact under development for this project, is done from the beginning of history, starting with an empty map that slowly becomes populated as time progresses. The impact of this approach to simulation allows the world to be truly defined by what occurred in its past: wars fought between nations in the past may affect the players ability to trade with others, artefacts of ancient history may be found by the player and can be stolen by enemies, the titular dwarves can even be seen engraving depictions of historical events or peoples in some cases.

This historical simulation system also has a direct impact on gameplay, a testament to the depth of the implementation. Some interesting examples of how the timeline of the world can affect gameplay are as follows:

* When starting a game, the player may choose their civilisation and what resources they will bring to their new colony. The resources available are defined by what the chosen civilisation is capable of – such as what metals they have available or what animals the nation has successfully domesticated.
* It is possible for a player to find themselves in a world that is missing a race, including as the playable dwarves, as a result of past events such as wars.
* The world is able to undergo changes in “age”, as defined by what peoples and monsters exist in the world at any given time. It is even possible for the world to enter a state in which all fantasy elements, such as dragons or monsters, have become extinct, which is referred to by the game as “The Age of Civilisation” (Dwarf Fortress Wiki, n.d.)

While this project includes many features applicable to this project, it is important to note the simulation serves as a backdrop to the main gameplay. Many liberties are taken to ensure the world is playable – a world without dwarves will still allow a player to start a colony and will simply spawn new people for the player to use. As such, while the game serves as an inspiration for this project, it has very little that can be used as material to work from. The following concepts highlighted are aspects which this project can gain insight from.

* + 1. World Generation

As described by the creator (Adams, 2009), the world generation method centres around elevation, with a location on the map being given a height and the sea level being defined as a result of the elevation of the world. This generated topographical map is then provided factors including temperature and rainfall – by which other factors such as river locations or biomes are built from. This approach to world generation is of particular interest, the concept of building a world by building layers of random content – elevation, temperature, rainfall – could be an appropriate method to be used by this project due to the relevance of these factors and their derivatives to the objective of the artefact.

The elevation generation seen in Dwarf Fortress makes use of a variant to the diamond-square algorithm (Adams, 2013) (Alain Fournier, 1982), the result of this being results that appear to be more continent-like than other algorithms in this field. Additionally, the developer of the game noted that the game did not use the more common Perlin noise method for world generation, citing it as looking “fluffy and rounded” (Adams, 2010). This assertion has implications for the direction of this project, as the diamond-square algorithm may serve as a better fit for the project than the original intended algorithm – Perlin noise.

* 1. Europa Universalis 4

Europa Universalis 4 is a game developed by Paradox Development Studio (Paradox Development Studio, 2013), which was released in August of 2013. The game presents a map of the real world as it emerges from the end of the medieval era, with gameplay spanning from the renaissance to the end of the age of revolutions. Unlike the aforementioned Dwarf Fortress, Europa Universalis 4 (Henceforth referred to as Eu4) places more emphasis on realism, with the map the game takes place on closely reflecting that of the real world in the time period a player chooses to start from. Additionally, the mechanics of the game strongly influence the human and computer players to act how a ruler in that time would have. This can be seen very clearly when reviewing the history of a game, in which certain real events will almost inevitably occur if the player does not intervene, such as the formation of the Russian empire, the uniting of the British Isles and the expansion of the Ottoman empire. Eu4 presents an interesting study into the field of simulating real world history, and examination into this media yields some interesting results.

* + 1. Provinces

First and foremost, the basic gameplay elements present within the game should be noted. The map in Eu4 is split into different “Provinces” – Polygonal areas representing the borders of a location, often defined by the existence of cities and population centres within the region. These provinces are the foundation on which the game is played – the nations a player represents own a certain number of provinces which represent the borders of the country, and the objective of the game can be said to be to build a large empire by taking as many provinces as possible, either through war or colonisation.

In addition to this, these provinces have a number of properties tied to them which determine how they interact with the gameplay. Some notable examples are the following (Paradox Development Studio, n.d.):

* Provincial development (Split between Taxation, Production and Manpower). This represents the infrastructure present in the area, higher developments corresponding to bigger cities which provide the player with more money, trade power and manpower. The development of a province also determines the value of the location in peace negotiations, the cost to the player to build an administration to the region as well as how the international community reacts to the annexation.
* Terrain. The environment of a province determines aspects such as the speed at which military units are able to travel through a location, the amount of supplies that can reach unit as well as the cost of developing the province. This provides a double-edged sword type scenario – a mountainous terrain is more difficult for an enemy to traverse and survive in, but the player will incur additional penalties for attempting to improve the infrastructure in the region.
* Culture. Each province has a culture that represents the peoples within it and occupying nations will find additional difficulty in attempting to maintain control of provinces with cultures that they do not support. Additionally, the player and the artificial intelligence is rewarded in various ways for occupying locations with their own national culture, incentivising the formation of borders based upon cultural boundaries.

The provincial system presented by Eu4 may be an aspect to consider in the design of this project. Splitting the world up in this way allows for the mapping of populations and population centres onto the model and may provide some advantages in organising the artificial intelligence of the simulation. As the system presented in Eu4 is designed to work on a pre-determined map, some adjustments need to be made to fit the random nature of the artefact.

* + 1. Artificial Intelligence

The Artificial Intelligence of Eu4 is largely classified data belonging to the development studio, however some things about its inner workings are known. As previously mentioned, the computer players will often prioritise the uniting of their culture, but this is additionally modified by aspects such as relations with foreign nations, the difficulty of taking provinces and the decision making of the rulers in charge of each nation.

A good example of how relations impact the decisions made by the AI can be seen in interactions between Spain (Known in the timeframe of the game as Castille) and Portugal. Both Spain and Portugal belong to the same “Iberian” culture group, and yet conflicts between them are minimised. This is partially due to the opinion modifier each country has on the other, citing historical alliances, meaning the two often prioritise other conquests rather than turning against one another.

In terms of rulers, computer-controlled nations will be designated a ruler “personality”, which is changed when a leader abdicates or dies. These personalities impact the actions nations will take during that leader’s reign, with militaristic rulers focusing on conquest and military expansion, and administrative rulers working instead to improve their economic condition. This creates a believable feeling of an evolving diplomatic landscape – players may find that once passive nations will suddenly become aggressive as a militarist takes charge, or former threats become minimised by a ruler less interested in war.

Particularly due to its ability to constantly change the diplomatic landscape of the world, ruler-based artificial intelligence is to be considered when determining the simulated history aspect of the development cycle. This system allows for great rises to power as well as falls due to negligence on the part of the ruler, which has been seen throughout real-world history extensively.

* + 1. Random New World

The first purchasable update to Eu4, named “Conquest of Paradise” (Valve Corporation, 2014) , introduced a feature referred to as “Random New World”, advertised as the ability to generate an entirely new continent as a replacement for the Americas. This meant players would have the ability to explore a random map and build a new colonial empire as they explored new terrain. Upon release, this particular feature was met with mixed reception – partially as a result of the algorithm used for the random new world generation.

The generation mapping makes use of set-piece based method – in which the newly generated map is built from a number of premade assets put together to make a map. This method has seen successful usage in games like Spelunky HD (Mossmouth, 2013) and The Binding of Isaac (Valve Corporation, 2011), however this method reflects poorly in a geographic environment. The set pieces present in “Conquest of Paradise” are often individual islands, rather than the large landmass present in the real-world Americas, and veteran players will quickly come to recognise the set pieces as they appear – removing the charm of the pseudo-random geography. Additionally, as the sets are human made, they often reflect themes or shapes – such as the set piece that resembles the corporate logo of the Paradox Development Studio – which does not create a believable environment for the game to be played upon.

Unlike the other aspects mentioned in the discussion of this product, this random generation method should be avoided, as it does not suit the attempted goals of the random world generation feature of this project. As an extension to this, the use of any predetermined structures, such as the polygons of provinces, should be avoided unless necessary – as they represent many of the same issues discussed above.

* 1. Sid Meier’s Civilization 5

Another game that has some relevance to this project is the 2010 Firaxis game Sid Meier’s Civilization 5 (Firaxis, 2010). In this game, the player is prompted to choose a historical nation and leader, before being presented with a hexagonal world map for them to build a new nation upon. Other nations, led by civilisations and leaders that the player may have selected, exist on this map concurrently, and the nations of the world will compete to reach a number of arbitrary “Victory” conditions. This game is far more gameplay focused than titles like the previously mentioned Europa Universalis 4, forgoing much realism in favour of creating an entertaining product. This would largely make it inapplicable to this artefact, which focuses almost entirely on realism, aside from the inner workings of the AI within the game.

* + 1. Artificial Intelligence

Each leader in Sid Meier’s Civilization 5 has a number of integer values representing their focus towards different tasks – set on a scale of 1 to 10 (With one glaring exception) – this in turn influences the actions the civilization in question will take. For example – the nation of France, led by Napoleon Bonaparte, has a “Loyalty” score of 3 – representing the real-life actions of the man, who was notable for violating various treaties and acting aggressively towards former allies. These values in turn then provide weighting to the decisions of the computer players, allowing them to act in pseudo-random ways which reflect the personalities assigned to them. This system allows for the nations of the world to act as if they have their own personalities and goals, and the actions of certain leaders can be predicted based on these hidden values.

This works as an extended implementation of the “personality” system present in Eu4. Whereas the Eu4 system works to provide a ruler a focus in areas such as diplomacy or economy, the Civilization approach instead provides weighting to specific areas – allowing for the characters within the game to have more of a character. This could see use in the artefact described in this document, with random rulers receiving random values that weight decision making. This, in addition, could permit the rulers to make mistakes – a ruler with a high weighting towards war could start conflicts with larger nations, potentially having significant consequences on the nation as a whole. The ability for a leader to make mistakes is one that should see implementation in the simulation, as misguided decisions have changed history countless times.

* 1. Iron Age

Iron Age (Gorman, 2018) was the working title of a previous attempt at a concept similar to that of this artefact conducted by myself in 2018. The software produced as a result of this endeavour was of amateur quality and did not meet the technical expectations it had set out to achieve, but yet managed to display two aspects relevant to the current artefact – World generation and time simulation. Despite the flaws of this project, some valuable research can be gained from investigation into the inner workings of this system.

* + 1. World Generation

Throughout development, Iron Age underwent no less than three different implementations of world generation algorithms – each uniquely designed to fit the project.

The first, referred to internally as the “Snake method”, involved a tile-based approach, in which a random index would be selected in a two-dimensional array as a “Snake”. This “Snake” block would then expand to an adjacent index, which would then repeat the process, expanding to another index location. Once this process was repeated a number of times defined by the parameters of the generator, each existing “Snake” block would generate 3 “land” tiles in each orthogonal direction, before becoming a “land” tile itself. This method allowed for the generation of long continents, as well as permitting the generation of features like inland seas and coves. However, in the end this method was not used due to its unpredictability and tendency to develop unrealistic geography.

The second method, the “Fill method”, was developed and rejected in less than a day. The concept was simple, populate a two-dimensional array with a number of tiles in random indexes with a number representing their elevation, then populate all tiles in a 3x3 area around said tile with a random modification to the elevation of its creator. Eventually this would produce a mountainous terrain map which would then be designated as sea level below a certain elevation, corresponding to the roughly 70:30 ratio of sea to land seen in real life. This system, while similar to those in use in similar projects, was not implemented in a way that benefitted from the strengths of elevation-based terrain – the nature of the expanding elevations meant that the landmasses produced were often of the same shape, and constructs like islands or inland seas simply did not appear.

Finally, the implemented “Continental method” was inspired by the real-world separation of Pangea into the modern continents. The model would produce a circle with a radius almost equivalent to the height of the map, which would be entirely filled with land tiles. The outermost tiles of the circle would then become “Active” tiles, which could move anywhere within a 3x3 area – switching the specified index to an active tile and replacing the previous tile with an ocean tile. These active tiles would then propagate simultaneously, creating random lines in the land, until they reached a point where there were no possible land locations to “jump” to in the 3x3 Area. Once all “Active” tiles had been removed, a flood fill algorithm would be utilised to identify each landmass and place it randomly in the ocean. This algorithm produced favourable results, producing random continents with believable proportions.

For this artefact, none of these methods should be used, as despite the relative success of the “Continental method” there were numerous flaws that could be alleviated by the use of other algorithms. Additionally, the continental method as implemented in Iron Age was particularly inefficient, using far more time and processing power than would typically be necessary for a world generation algorithm.

* + 1. Simulation & Artificial Intelligence

As discussed, Iron Age attempted to meet many of the same objectives as outlined in this artefact, including the development of nations across time. The simulation aspect of Iron Age was flawed in many ways, first and foremost in its scalability, a mistake which cannot be repeated in this artefact.

Iron Age implemented a system in which all nations generated a certain amount of “military units” per season, the amount of which would be modified by the technology of the owning civilisation, the number of tiles owned by the civilisation and the number of resources in the capital of the owning civilisation. These units would continuously generate as time progressed and would only be expended when the computer players attempted to take land from another nation. This resulted in an inevitable overflow in which a nation would exceed the bounds of the integer data allocated as their military size – an issue that was compounded by the already extensive use of memory by the software.

Additionally, the artificial intelligence implemented for Iron Age was very simplistic, and the only concerns of nations revolved around taking more land – with little regard for any properties of the land in question. This was sufficient for the small scale of Iron Age and its usage as an experiment into simulation but prevented the system from depicting a believable world. Additionally, the nations of the world fit a rigid decision-making structure – only declaring war if they had a numerical advantage over their competitor of above a certain constant value. This prevented the nations from taking risks, or developing any real form of personality, an aspect which will be emphasized in the development of this project.

* + 1. Tile and National Names

As a final note on Iron Age, there is one method that may see recreation for the purposes of this artefact, notably in the naming system for locations. Upon generating a world, the user would be prompted to select a method of name generation – either by random selection or random generation. Upon selecting random generation, the system would create a series of new location names based on existing names of cities.

This was done by first randomly selecting an index within the dataset containing the names of various cities, the first number of characters from this name would then be stored in memory. The algorithm would then find another random index, and compare the last stored character to the character in the same position in the character array (For example, if the memory stored the words “Lond”, a comparison against the city of “Leeds” would return true, as the fourth character is shared), after which the mechanism would pull a number of characters from this word to append to the original memory (In the listed example, this would produce “Londs”). This process would continue until the system reached a blank character or no names that fit the condition were found (after an iteration through the source material starting from the first failed comparison). The produced name would then be stored with an index representing the location it applied to.

While this system had some flaws, the results given were often believable location names that fit the simulation well, as such it is possible that this algorithm may be reworked to fit this artefact – though with better efficiency measures and potential quality checking systems implemented alongside it.

1. World Design

As a result of the collated information and comparison in the literature review, the following design plans have been developed to specify how the system itself will function – in particular what features will exist and what algorithms will be used where appropriate.

* 1. World Map
     1. World Generation

The intended algorithm for the world generation was originally the Perlin noise method, but due to the results of the literature review the “Diamond-square algorithm” has been identified as a better fit for the goal of generating a landscape – and will therefore take the place of Perlin noise in this area of the system. This is not to say that Perlin noise will see no use in the project, as review into systems like dwarf fortress have identified the necessity of different “layers” of randomly generated data – it is not enough to simply create a landscape. The following “layers” will need to be produced in any single use of the map generator:

|  |  |  |
| --- | --- | --- |
| **Layer** | **Layer Name** | **Algorithm** |
| 0 | Elevation | Diamond Square |
| 1 | Temperature | Random gradient from the equator |
| 2 | Rainfall | Perlin Noise |
| 3 | Flora | Perlin Noise |

This in turn should make the following adjustments and additional values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Order** | **Name** | **Relevant Factors** | **Description** |
| 0 | Land/Sea Adjustments | Elevation | All values over the 70th percentile of elevation will be defined as land tiles, while all below will be defined as sea. All values relating to the sea locations will be discarded, and any following operations will only occur on the land tiles. |
| 1 | Mountain Adjustment | Elevation, Temperature | The top 20% of elevation values will have their temperature values lowered proportional to their deviation from the 4th quintile elevation. |
| 2 | Forestry | Flora, Rainfall | Locations are given a forestry level based on the overlap between above-average flora values and above-average rainfall values. |

All these collated factors will then be used to specify a biome type for each pixel, using a system by which the locations “fit” value to a biome type is a number between 0 and 1 that is calculated based on the weighting each biome assigns to a specific property. While exact values for weighting are not yet set, the rough impact each property will have on the “fit” value can be seen in the table below. For each property, the difference from the median value for the field is taken as the input and multiplied by the weight (specified here as either high or low) – therefore a negative difference will count as the opposite for the weighting (for example, a biome with negative forestry rating will return a negative number for a highly forested region, but will produce a positive value for a less-forested-than-average region)

|  |  |  |
| --- | --- | --- |
| **Biome** | **Value Weighting** | **Colour** |
| Temperate Forest | Forestry (High Positive Weight)  Temperature (Low Negative Weight) | Dark Green |
| Tropical Forest | Forestry (High Positive Weight)  Rainfall (Low Positive Weight)  Temperature (High Positive Weight) | Very Dark Green |
| Taiga | Forestry (High Positive Weight) Temperature (High Negative Weight) | Light Green |
| Grasslands | Forestry (Low Negative Weight)  Rainfall (High Positive Weight)  Elevation (Low Negative Weight) | Green |
| Savannah | Forestry (Low Negative Weight)  Temperature (Low Positive Weight)  Elevation (Low Negative Weight) | Orange |
| Tundra | Forestry (High Negative Weight)  Temperature (High Negative Weight) | Very Light Green |
| Desert | Forestry (High Negative Weight) Rainfall (High Negative Weight)  Temperature (High Positive Weight) | Yellow |
| Mountain Range | Elevation (High Positive Weight)  Forestry (High Negative Weight) | Grey |
| Forested Plateau | Elevation (High Positive Weight)  Forestry (High Positive Weight) | Green-Grey |
| Shrubland Plateau | Elevation (High Positive Weight)  Temperature (High Positive Weight)  Forestry (Low Positive Weight) | Orange-Grey |

* + 1. Provinces

Once all the geographical information is produced, the world generation algorithm can move towards the next stage of human history – the dispersion of humanity across the world. In this stage, the existing map will be divided into a number of polygons – each representing the geographical boundaries of a population of people, the rough borders at which the residents of a location live their lives.   
  
The first stage of this process is to divide the map into a number of equally sized squares, each of these squares that contain land is then split into two triangles, randomly selecting the direction of the division (from top left to bottom right or from top right to bottom left). Each of these locations is referred to as a “chunk” and represents a fraction of a province. Each chunk will then store the following information based on the average values of its containing pixels: Elevation, Temperature, Rainfall, Forestry and Biome. All chunks will then store an enumerable referring to their relative position in the set of all chunks.

When all the chunks have generated, a random chunk will be randomly selected from the set of all chunks to form the first component chunk of a “province” – the base measurement of land to be used by the simulation. The first chunk in the province will return its biome which will be polled for its “spread” value – a rough indication of the population density of the region (biomes with more dispersed populations, such as those of tundra or desert regions, have a high spread value. Those with more concentrated populations, like those of forested regions, have lesser spread values).

Upon receiving the spread value as dictated by the province, the province will then proceed to spread across multiple chunks – choosing any chunk with two or more connecting vertices to be added to its components (upon which the adjacent chunks to the new component chunk will be accessible by the province) and repeating this process for as many times as the spread value allows. A province will choose the next chunk to be appended to its components set by randomly selecting a chunk from a compiled list of all valid chunks, with the following rules determining what can be considered a valid chunk:

* Only chunks with two or more connecting vertices are able to be added as a component chunk. When the set of all valid component chunks is developed, the software will check for any repeated chunks within the set. If the set contains multiple of the same chunk, all chunks with less instances than the highest number of repeated chunks will be discarded.
* Provinces that start in an ocean biome (dictated by having no non-ocean tiles in the chunk boundaries) may only claim adjacent chunks of other ocean biomes. Similarly, land biomes may not claim ocean biome chunks.
* Any chunk appended to a province (either the first chunk or any connected chunks) may not belong to any previously generated chunk – preventing any overlaps in claims.
* A province with no valid adjacent chunks will cease appending new members to the set of component chunks regardless of the remaining spread value.

The generation of new provinces will proceed until all chunks generated are a member of a single province. The process of generating provinces allows for the world to be split into polygons according with size dependent on their biome – creating an environment split between the populations that live upon it.

* 1. World Properties

As mentioned in the prior segment – each location on a map will be sorted into different “provinces” which are the basic “unit of measurement” upon the map. A province represents not only the borders of a location but the borders of the people that inhabit it – and what conditions they live in. Like the chunks and the individual tiles on the map, each province will contain its own list of properties that describe it – first and foremost the biome defining properties (Elevation, Temperature, Rainfall and Flora) as well as the biome itself.

Each province will also require its own unique name to serve as a descriptor of the lands. This name generation algorithm will use the algorithm described in the literature review for “Iron age”[[1]](#footnote-1), with variations to improve performance and fit towards the project. This means the project will need to implement a dataset of numerous city names to allow for this property generation. Unlike the referenced software however, this procedure will be expanded to provide the ability to generate names for other constructs, such as oceans. In addition, the performance of this algorithm will be amended by rebuilding the algorithm from the ground up, using the concept rather than the code to allow for modifications to be made which produce a better performing product.

Additionally, each location will require its own “Culture”, a property that will be shared across a large variety of different provinces. A culture refers to a group with similar principles and ideas, effectively representing a “block” of people. Provinces in a similar area should share the same culture, and each culture should be provided a name to identify the people within, much like how the name of a location is generated.

Finally, each province should contain a property defining the starting population metric within its borders – using the previously generated parameters to do so. These population metrics will specify the size of the capital city within each province, either as a “Village”, a “Town”, or a “City”. While the population metric is liable to change throughout simulation, setting a baseline for these values should allow for a better start to the simulation.

1. World Generation

The world generation of the artefact, Juris-Civilis, met most of the goals outlined for the project within the allocated deadlines, but presented some unexpected challenges throughout development which necessitated changes in direction and procedure to be made. While most of the implemented features remained faithful to the original designs, some (with much deliberation) were modified to better fit the technical and creative requirements of the system.

First and foremost, the previously undefined parameters of the system – particularly the size of the map were determined. After testing, the best fitting width and height were determined as 6000x4000 (in pixels) – creating an image with a height to width ratio that both resembled that of a Mercator projection map as well as giving space to the map generator to operate effectively. While implementing all subsequent procedures, special consideration was made towards keeping all operations designed for different map sizes – with the intention of potentially permitting different map sizes and ratios in the future.

For the generation of continents, the originally specified diamond-square algorithm was discarded – replaced by a Perlin noise algorithm making use of octaves to generate jagged terrain. This algorithm, when compared with the previously mentioned method, demonstrated a better fit towards the direction of the artefact. Due to this change, review was made into the relevance of other described algorithms within the design plans – with little changing in aspects other than that of the temperature generation. While originally intended to use purely a curved line algorithm, the implemented system uses a mixture of a Perlin noise generator as well as a gradient function from a randomly generated line across the centre of the map. This change better reflects the variance in heat across the world, which is not a consistent line but rather a more varied mesh.

When the first results of generating terrain were made it became apparent that the noise generation procedures were liable to produce continents across the map borders. In most modern maps, the map is orientated in a way that cuts the map borders through the sea, rather than through land – this change allows maps to be more easily readable, as consideration does not often need to be made into which side of the world a location falls under. This meant that for the map to properly resemble a Mercator projection map, either the map had to orient itself in a way that cut through the seas, or the map had to generate in a way that did not step into map border territories. To fix this issue, a new algorithm was produced to reduce the intensity of generated properties when they approached a certain boundary of the map. The values of properties within a certain distance from a map boundary were divided by greater values as they approached the border – allowing for a sloped decline in values. While this procedure was originally only designed for the elevation, it also found use in generating the polar regions of the map, with the temperature of a location being rapidly decreased if it fell within designated polar regions.

One unexpected trial of development came in the form of memory usage – which first appeared after the implementation of the four primary world generation factors (elevation, temperature, rainfall, flora). In accordance with the original design plans, each pixel of the map stored its own integer values for the four properties. With the map size of 6000x4000, this resulted in a set of twenty-four million elements storing 16 bytes of memory each. Additionally, the next step was to be the compilation of four sets containing a sorted list of each property to determine the decile values for each property. This procedure took too much memory to process, and therefore was modified in two ways; Firstly, the pixel class was changed to store only enumerators referring to the relative position of each property in relation to the set (in essence, instead of storing elevation as a number, it stored the value as “Low” or “High”). The enumerators generated by this procedure are as follows:

|  |  |
| --- | --- |
| **Property** | **Enumerator / Decile** |
| Elevation | |  |  | | --- | --- | | Low | Less than 6th decile | | Medium | Between 6th decile and 9th decile | | High | Greater than 9th decile | |
| Temperature | |  |  | | --- | --- | | Low | Less than 3rd decile | | Medium | Between 3rd and 7th decile | | High | Greater than 7th decile | |
| Rainfall | |  |  | | --- | --- | | Low | Less than 4th decile | | High | Greater than 4th decile | |
| Flora | |  |  | | --- | --- | | Low | Less than 5th decile | | High | Greater than 5th decile | |

Secondly, the decile calculation was changed from using a median based procedure to using a range-based formula, removing the need for the sorting of the properties lists entirely. The new function used the following calculation to determine the rough value for each decile.

These changes, while making the results of the generation algorithm less accurate and refined in places, consistently cut down memory usage from the software by a fourth – a huge change that made up for the minute decrease in quality as a result of the changes. In addition to this, the described property adjustments[[2]](#footnote-2) were cut from the function entirely – as the switch from exact integer values towards enumerators demonstrated a lack of need for these changes – as a result of this, the forestry property was cut entirely, being replaced by the previously generated flora values.

* 1. Biomes

As these changes reflected large modifications to the design, the biome system, previously dependent on the replaced integer system, was overhauled too. These changes were mostly to better fit the new enumerator variables into the procedure, but these changes largely made the map produce better results overall. In the new system, biomes each had a “expected” enumerator value for each property, as well as an integer value which scored their “fit” for each biome. For each matching value, the “fit” score would increment by one. The highest scoring biome would be applied to the pixel. Two exceptions existed for this method however – the elevation of a province applied for ten “fit” points rather than one – ensuring ocean biomes and mountainous regions would be set appropriately – and each biome may have an enumerator value of “NA” which would automatically apply half-a-point of “fit” score to the biome score. The following table describes the newly formed biome properties.

|  |  |  |  |
| --- | --- | --- | --- |
| **Biome** | **Value Weighting** | **Colour (RGB, Normalised to 0-1)** | **Provincial Spread** |
| Ocean | Low Elevation  NA Temperature  NA Rainfall  NA Flora | (0.04,0.08,0.58) | 400 |
| Temperate Forest | Medium Elevation  Medium Temperature  NA Rainfall  High Flora | (0.01,0.39,0) | 16 |
| Tropical Forest | Medium Elevation  High Temperature  High Rainfall  High Flora | (0.06,0.34,0.05) | 36 |
| Taiga | Medium Elevation  Low Temperature  NA Rainfall  High Flora | (0.06,0.22,0) | 36 |
| Grasslands | Medium Elevation  Medium Temperature  NA Rainfall  Low Flora | (0.02,0.54,0) | 24 |
| Savannah | Medium Elevation  High Temperature  High Rainfall  Low Flora | (0.78,0.55,0.15) | 16 |
| Tundra | Medium Elevation  Low Temperature  NA Rainfall  Low Flora | (1,0.78,0.78) | 48 |
| Desert | Medium Elevation  High Temperature  Low Rainfall  Low Flora | (0.77,0.61,0.23) | 36 |
| Mountain | High Elevation  Low Temperature  NA Rainfall  Low Flora | (0.5,0.5,0.5) | 8 |
| Forested Plateau | High Elevation  NA Temperature  NA Rainfall  High Flora | (0.5,0.5,0.5) | 8 |
| Shrubland Plateau | High Elevation  High Temperature  NA Rainfall  Low Flora | (0.5,0.5,0.5) | 8 |

Map

Description automatically generatedThe completed biome system is able to produce results depicting a believable continental structure and demonstrates many geographic principles such as the concentration of temperature based on distance towards polar regions or the equator. The implemented biomes create diverse ecosystems that are later used in the simulation to demonstrate how the factors that make up a biome can affect the development of a world.

Figure 1 A world generated using the world generation algorithm

Once the world generation procedure finishes, the software quickly moves onto splitting the world into provinces. The provincial system has no variance from the way it was designed – splitting the map into right angled triangles and then connecting them to form provinces. As shown in the biomes table above, each biome has an appropriate spread value which dictates how far the borders can spread across the world map. These spread distances are determined by the initial starting location of a province, but once generated, the properties of a province are determined by the most prevalent of each property in each chunk – meaning a province with 6 chunks, with 4 high elevation chunks and 2 medium elevation chunks will be considered a high elevation province.

1. World Properties

After generating the world map, the software begins the procedure of generating values for each province, allowing for the simple polygonal locations to have uniquity, giving more credence to the idea that these shapes represent a group of people and their lives. In order to better represent these properties on the map, new “map modes” were implemented, allowing the map to show users different relevant statistics at a single glance. These map modes can be selected at any time using the dedicated map mode panel.

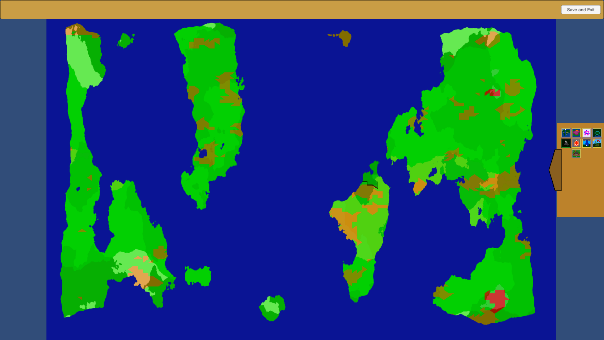
The first property to be implemented was the inclusion of hills and craggy terrain. The original map generation stores the elevation of a location as a trinary set – “Low”, “Medium” or “High” – with low referring to anything below sea level, medium as anything on land, and high as any mountainous location. This initial procedure generated relatively flat land, with only minor inclines in the form of mountains and plateaus. To fix this, the hills procedure first sets all “Medium” elevation land to “Low” (Excluding oceans, which fall under a new property, “NA”) before selecting a random assortment of these “Low” provinces to be incremented to “Medium”. The implementation of hills provides a new avenue for the eventual decision-making procedure to consider – as heightened terrain provides a defensive advantage to any owner of the province.

Figure 3 An elevation map. Showing green land as flat land - yellow land as hills and red land as mountains.

Figure 2 An example of the map mode window. Each icon representing a different displayable property on the map. In this screenshot, the elevation map mode is selected.

* 1. Cultures & Names

Secondly, each province on the map was given a unique name – generated randomly using the naming procedure introduced in the “Iron Age” project. By selecting random amounts of letters from a real city name, and then appending another random number of letters from a city name with a letter in the same position – realistic provincial names can be created to give each location on the map a unique “capital” to represent the location as a whole. The data set from which these names are pulled contains a listing of over one hundred and thirty thousand unique city names from across the globe. This procedure, while sometimes tending to generate inappropriate outputs, often produces realistic sounding city names, with the matching of characters property meaning that cities from similar cultures and languages are often paired together. Some examples of names produced by this algorithm are the following:

* Krasonvi
* New Marti
* Attastia
* Endalu
* Bajakh
* Vantelier
* Ameraya

This algorithm was then built upon to produce two variants – the cultures procedure and the ocean names. In the terms of the simulation, a culture refers to a group of provinces that share similar ideas and ethics. In the simulated environment, provinces that share a culture with a nation are the main focus of a nation – as it seeks to unify its people and handles complex relations with potential allies or rivals.

Map

Description automatically generated  
The culture procedure uses a variant of a flood-fill algorithm. At the start, a number of random provinces are selected to form new cultures. One of these provinces is then selected to spread its culture to all adjacent provinces that do not yet have a culture. Once this is completed, the procedure starts again, a random culture is selected, and a random province with adjacent provinces without a culture is selected within that culture to spread to all adjacent provinces. This procedure repeats until all cultures have no more applicable adjacent provinces, causing the software to check that all provinces have been granted a culture. If this condition is not met, a new number of provinces with no assigned culture are granted new cultures to spread to their adjacent – repeating this procedure until every province has a culture assigned to it, no matter how small said culture may be. The results of this represent a varied world of different groups – mainland provinces will often belong to large contiguous culture blocks, whereas islands will often have cultures that extend only to the borders of the island.

Figure 4 A map of the different cultures on a world map

These cultures, like the provinces that encompass them are granted names. This name generation algorithm, as previously mentioned, uses a variant of the province naming procedure. A dataset included with the software contains a list of roughly seven-hundred different adjectives to describe cultures, nations and ethnic groups. Each name in this set is then reversed before the prior described name generation procedure is applied – ensuring that the suffixes are maintained. When the desired number of culture names are generated, the names are applied randomly to each culture. As the sample size of names is relatively low, there is a high risk of duplicate names, therefore the algorithm queries the set of generated names for duplicates before appending any new culture names to the set. Some examples of culture names generated are the following:

* Ippian
* Digarhi
* Canese
* Lusian
* Lilandian
* Aquenian

Finally, each ocean is also provided its own name, using the same algorithm as the culture naming procedure but with a different dataset. The dataset in question in this instance includes almost five hundred unique eponymous adjectives, describing concepts such as philosophies and religious figures. This produces a set of names that are noticeably distinct from cultures and lean more towards concepts rather than descriptions. After this procedure is complete, each name is provided a suffix to that refers to its position relative to land, creating names like the following:

* Ersonian Ocean
* Censian Gulf
* Vaudian Sea
* Bertite Ocean
* Aurian Ocean
* Ysonian Sea
* Nesian Cove
  1. Population

After generating these properties, the arguably most important metric can be determined – the population of a region. This property will later determine aspects like when nations will rise and fall, and so must incorporate most of the properties generated to create an estimate for where people are most likely to live. For this, there are five values used – Culture size, Elevation, Temperature, Rainfall and Flora.

Each of these properties are given numeric values to allow their usage in a formula to determine a “score” for the population of a region. Culture size, referring to the number of provinces on the map that adhere to a specific culture, is given a decimal value from 0 to 1, normalized according to the highest culture size and lowest culture size on the map. A provinces elevation is given a value of -1 if in the lowlands, 0 if on a hill and 1 if on a mountain, likewise, temperature receives a -1 for a cold climate, 0 for a temperate climate and 1 for a hot climate. For the two Boolean properties, rainfall, and flora, they are each provided a 0 for a “Low” value and 1 for a “High” value. These numbers are then used in the following equation:

This equation makes use of each property in a way that reflects the way a property would impact the population of a region – for example, the temperature value (T) is used with the sec function, providing a lower value for a 0 or 1 temperature value, but a higher value for a median temperature – allowing populations to generate in locations with less extreme climates. The produced results of this demonstrate a world defined by its surroundings, with populations mostly generating in close proximity to one another.

Map

Description automatically generated

Figure 5 A world population map. In this world, civilization has largely centered around the upper right corner of the map due to its moderate temperature, high rainfall, and high forestry. Other populated regions can be seen scattered across the map

* 1. Saving

Once a world has been generated, it is automatically saved to a user’s disk in the directory from which the artefact is being executed. If one does not already exist, a “Saves” directory will be generated to contain the worlds generated – which each receive a name such as “World0” – with the suffix number incrementing based on how many worlds exist in the save’s directory. The save data folder contains a number of relevant data to be loaded in the simulation stage, most notably the provincial properties and component vertices and the world map image.

* + 1. XML files

Each save file within the document is written in an XML format – with nodes representing concepts such as provinces and cultures. When first generating a world, three files are written to – “World.sav”, “WorldData/Provinces.xml” and “WorldData/Cultures.xml”. The first file, the world save file defines the basic properties for a world, such as the height and width of the map, and its data will be loaded first to ensure all world configuration settings are maintained.

The provinces xml file contains all the relevant province object data – including its name, ID, properties, culture, and mesh vertices, allowing for the province to be loaded in the same format it was originally generated in. When loaded, the map procedure will automatically reconstruct the province meshes from the vertices provided in the correct order to ensure the vertex faces align.

The cultures xml file contains all relevant data to cultures, such as the name and ID. Provinces only store the ID for their culture, and therefore it is imperative to load the list of cultures to serve as a reference for this data.

* + 1. PNG files

Within the save directory, there are two image files – “Map.png” and “Mask.png”, each with their own important purpose to help with the presentation of the simulator. Map.png stores the map texture, the backdrop for all the provinces to be rendered in front of, while Mask.png stores only the ocean texture. This masking image is placed in front of the provinces, creating an effect in which the provincial borders fit to the land borders, while still maintaining their initial interaction hitbox. By using both these image files, the map can be stored and loaded in a way that presents its best aspects.

1. Bibliography

Adams, T., 2009. *Dwarf Fortress Talk #4* [Interview] (3 November 2009).

Adams, T., 2010. *Nullpointer Dwarf Fortress Interview* [Interview] (1 July 2010).

Adams, T., 2013. *Dwarf Fortress Talk #21* [Interview] (23 May 2013).

Alain Fournier, D. F. C., 1982. Computer rendering of stochastic models. *Communications of the ACM,* 25(6), pp. 371-384.

Dwarf Fortress Wiki, n.d. *Dwarf Fortress Wiki - Calendar.* [Online]   
Available at: https://dwarffortresswiki.org/index.php/DF2014:Calendar

Firaxis, 2010. *Civilization 5 Home Page.* [Online]   
Available at: https://civilization.com/en-GB/civilization-5/

Gorman, J., 2018. *Github - Iron Age.* [Online]   
Available at: https://github.com/JaVonox/Iron\_Age

Lorenz, E., 1972. *Does the flap of a butterfly’s wings in Brazil set off a tornado in Texas?.* Cambridge, Massachusetts, s.n.

Mossmouth, 2013. *Spelunky World.* [Online]   
Available at: https://spelunkyworld.com/

Paradox Development Studio, 2013. *Paradox Plaza.* [Online]   
Available at: https://www.paradoxplaza.com/europa-universalis-iv/EUEU04GSK-MASTER.html

Paradox Development Studio, n.d. *Europa Universalis Wiki.* [Online]   
Available at: https://eu4.paradoxwikis.com/Province

Tarn Adams, Z. A., 2006. *Bay 12 Games.* [Online]   
Available at: http://www.bay12games.com/dwarves/

Valve Corporation, 2011. *Steampowered Store - The Binding of Isaac.* [Online]   
Available at: https://store.steampowered.com/app/113200/The\_Binding\_of\_Isaac/

Valve Corporation, 2014. *Steampowered Store - Europa Universalis IV Conquest of Paradise.* [Online]   
Available at: https://store.steampowered.com/app/241369/Expansion\_\_Europa\_Universalis\_IV\_Conquest\_of\_Paradise/

1. See segment 2.43 [↑](#footnote-ref-1)
2. Referring to the design document property adjustments [↑](#footnote-ref-2)