UNIVERSIDAD
COMPLUTENSE
MADRID

Grado en Desarrollo de Videojuegos

Trabajo de fin de grado

Generación procedimental de asentamientos humanos en mundos 3D (Procedural Generation of Human Settlements in 3D Worlds)

Autores:

Miguel Ángel Gremo Alcocer
Borja Núñez Béjar
Gabriel Aramis Sardaneta del Collado

Director:

Prof. Dr. Carlos León Aznar

Facultad de Informática Universidad Complutense de Madrid
2019/2020

Madrid, Junio 2020

Índice

Resumen

La generación procedimental es importante hoy en día en la industria del videojuego, ya que agiliza la creación de nuevo contenido permitiendo ahorrar recursos y dando al desarrollador la opción de centrarse en otras labores. En este contexto, se ha desarrollado una herramienta con el objetivo de generar asentamientos humanos virtuales, teniendo en cuenta que todos los elementos del mapa mantengan coherencia entre sí y resulten atractivos según el criterio de evaluadores humanos.

Para la representación de los asentamientos se ha creado un mod del videojuego *Minecraft* que permite ejecutar los comandos necesarios para modificar el entorno. Se ha escogido esta plataforma porque permite generar mapas aleatorios coherentes con distintos biomas y superficies, gracias a ello se pueden probar los algoritmos en una gran variedad de situaciones y comprobar su adaptabilidad.

Las bases y criterios que se contemplan en *The GDMC Competition (Generative Design in Minecraft)* han servido como referencia para el diseño de los asentamientos y la realización de pruebas con el fin de evaluar sus características. La versión implementada en este trabajo se ha enviado a la edición de 2020 del concurso.

El resultado final del proyecto cumple los objetivos planteados pues permite generar asentamientos en *Minecraft* que replican la estética y organización de las civilizaciones humanas.

Abstract

Nowadays, the procedural generation is important in the video games industry since it accelerates the creation of new contents saving resources and giving the developers the opportunity to focuse on other tasks. In this context, a tool has been developed in order to create virtual settlement in Minecraft bearing in mind that every element from the map have coherence with each other and turn to be attractive to people.

For the settling representation, a mod of the videogame named Minecraft that allows executing the necessary commands to modify the environment has been created. This platform has been chosen because it allows the creation of random coherent maps with different biomes and surfaces. Thanks to that, the algorithms can be executed in a large variety of situations and verify its adaptability.

The bases and criterion contemplated *in The GDMC Competition (Generative Design in Minecraft)* have served as a reference for the design of the settlements and the test carrying out in order to evaluate their characteristics. The version implemented in this work has been sent to the 2020 edition of the competition.

The result of the project accomplishes the raised objectives as it allows the generation of Minecraft settlements that replicate the aesthetics and organization of human civilizations.

1. Introducción

La generación procedimental es un recurso cada vez más empleado en la industria de los videojuegos. *(Frade, M., de Vega, F. F., & Cotta, C., 2012).* La capacidad de creación aleatoria aporta variabilidad a los juegos y reduce la carga de trabajo en casi todos los aspectos del desarrollo como la programación, el arte y el diseño. *(Shaker, N., Togelius, J., & Nelson, 2016)* Son estas ventajas las que han llevado a su uso en otras industrias como el cine, permitiendo generar rápidamente escenarios y modelos.

El propósito del estudio es el desarrollo de una herramienta que alivie esta carga de trabajo mediante la generación de asentamientos virtuales en *Minecraft (Persson, M., & Bergensten, J., 2011)*, aprovechando el terreno creado mediante el generador incluido en el juego, y teniendo en cuenta que todos los elementos mantengan coherencia entre sí y resulten atractivos, con el fin de lograr como resultado un mapa capaz de rivalizar con uno generado a mano.

El entorno elegido para el desarrollo del proyecto es *Minecraft*, uno de los máximos exponentes de la generación procedimental en videojuegos, tratándose de uno de los juegos más vendidos de la historia. *(Ames, M. G., & Burrell, J., 2017).*

Además de ser un *best-seller*, su modelo de generación de terrenos lo convierte en un campo de pruebas idóneo para desarrollar algoritmos de generación procedimental que requieran una base geológica compleja y realista.
A la hora de generar el asentamiento se va a utilizar un conjunto de algoritmos que realiza los siguientes procedimientos:

En primer lugar, se realiza el preprocesado del terreno, estos algoritmos se encargan de analizar el área de construcción y tratar el terreno con el fin de crear una superficie más consistente para el algoritmo de construcción.

Una vez se ha realizado la preparación del terreno se dará paso a los algoritmos de distribución de parcelas, estos eligen los lugares óptimos donde construir cada edificio.

Por último, una vez todos los edificios estén asignados, es el turno de la fase de edificación, en la cual se unen las distintas parcelas mediante caminos y se construyen los distintos elementos instanciando plantillas diseñadas anteriormente.

El trabajo se ha realizado con vistas a participar en *The GDMC Competition,* una competición anual en la que distintos participantes generan algoritmos para producir asentamientos de forma procedimental en *Minecraft*. El reto consiste en crear un algoritmo que produzca un asentamiento coherente en un mapa

aleatorio. Estos asentamientos deben ser adaptables al terreno, funcionales y tener una narrativa y una estética interesantes. *(Salge, C., Green, M.C., Canaan, R., & Togelius, J., 2018).*

1.1 Motivación del estudio

La generación procedimental de niveles es necesaria en la industria del videojuego, ya que cada vez aparecen más desarrolladores que quieren dedicar sus esfuerzos a otros apartados de sus proyectos como la narrativa o el desarrollo de mecánicas, por lo que esta herramienta les permite ahorrar tiempo en generar mapas para sus experiencias, prototipos o videojuegos. *(Khalifa, A., Perez-Liebana, D., Lucas, S. M., & Togelius, J., 2016)*

Al ser una industria en auge este tipo de herramientas se está empleando cada vez más en distintos ámbitos del desarrollo, ya que además del ahorro de recursos, la oportunidad de añadir un componente aleatorio a la creación de los mapas abre la puerta a nuevas posibilidades de diseño y aporta rejugabilidad. Entre los usos de esta técnica destaca la creación de elementos naturales como biomas, ecosistemas e incluso planetas y galaxias.

En concreto este trabajo se enfoca en la generación de asentamientos ya que el diseño de mapas es una de las partes más complejas del desarrollo de un videojuego.

En cuanto a la plataforma en la que se va a desarrollar la herramienta, se ha escogido el videojuego *Minecraft*, ya que, además de ser un *best seller* con una enorme comunidad, cuenta con un modelo de generación de terrenos realista que simular entornos geológicos complejos. Esto permite comprobar el comportamiento de los algoritmos en gran cantidad de situaciones. Por otro lado, su representación gráfica está basada en vóxeles, lo que facilita la construcción y permite gestionar de forma sencilla el mundo general al poder manejarlo como una matriz tridimensional compuesta por cubos.

Como se ha mencionado anteriormente, la comunidad está en continuo crecimiento, esto favorece el empleo de este tipo de herramientas ya que existen comunidades enfocadas a la investigación de la creación procedimental de mapas y ciudades para videojuegos, en ellas los usuarios comparten sus mods y semillas de los mundos que han generado, permitiendo al resto replicarlos y visitar lugares curiosos o de interés.

Destaca además el uso de este tipo de herramientas en el campo de la enseñanza. *(Ekaputra, G., Lim, C., & Eng, K. I., 2013).* Mediante el uso de mods de Minecraft

es posible generar entornos útiles a la hora de impartir materias como biología, física, química, geología, etc*. (Short, D., 2012).*

1.2 Hipótesis de partida

El objetivo de este trabajo es desarrollar una herramienta que genere asentamientos automáticamente que resulten coherentes, funcionales y visualmente atractivos a ojos humanos. Al ser la creatividad un componente fundamental no es posible medir los factores que hacen que un asentamiento cumpla estas características, por lo que se tienen que crear hipótesis partiendo de planteamientos subjetivos, empleando métricas que simulen el comportamiento humano para mejorar la evaluación y percepción del resultado final.

Para la creación de asentamientos es interesante además añadir un componente "humano" que le dé sentido al resultado final, para que los pueblos parezcan naturales y no generados de forma artificial. Basarse en los criterios para crear ciudades que utilizaría una hipotética civilización aporta coherencia a la estructura de sus elementos y valor narrativo.

Partiendo de estas bases se han planteado las siguientes hipótesis:

- Hipótesis de los focos: Si se establecen primero los focos de la ciudad y luego se construye el pueblo en función a la distancia a ellos, se obtiene un resultado más fiel al modelo urbanístico radial que se trata de simular.

- Hipótesis de la inclinación: Escanear previamente el terreno y priorizar las parcelas con menos inclinación da un resultado más coherente porque se construyen los elementos en zonas con menor desnivel.

- Hipótesis del ruido: Si se aplica un desplazamiento en base a un ruido gaussiano a la hora de distribuir las parcelas, se obtiene un resultado menos cuadriculado y más "humano".

- Hipótesis de los caminos: Si juntamos casas y luego juntamos esos conjuntos de forma recursiva hasta que solo quede un conjunto, cuyo centro sea el foco del pueblo, se obtiene una red de caminos ramificada que se asemeja de forma realista al modelo urbanístico radial que se trata de imitar.

- Hipótesis de la decoración: Eliminando previamente toda decoración existente en el mundo y colocándola una vez se han construido los caminos y edificios se obtiene un resultado más ordenado.

Las hipótesis anteriores se resumen en la posibilidad de utilizar algoritmos que simulan bases arquitectónicas históricas para generar un pueblo con coherencia y estética similar a uno diseñado a mano.

1.3 Objetivos

El desarrollo del trabajo viene marcado por el cumplimiento de los siguientes objetivos:

- Analizar las características que hacen que una ciudad resulte atractiva y coherente con la estética.
    - Realizar un estudio de los distintos modelos urbanísticos con el fin de generar asentamientos que sigan distribuciones urbanísticas coherentes.
    - Estudiar los elementos estéticos que caracterizan los asentamientos humanos para replicarla de la forma más fiel posible.

- Diseñar métricas que valoren distintos aspectos del asentamiento obtenido.

- Generar asentamientos coherentes a partir de mapas generados previamente por *Minecraft*, utilizando algoritmos que modifiquen sus elementos.
    - Procesar los desniveles del mapa para conseguir un terreno donde se pueda construir un pueblo sin afectar de manera drástica a la disposición original.
    - Determinar cuál es la mejor manera para distribuir los distintos edificios, atendiendo a distintos parámetros (inclinación del terreno, distancia a puntos de interés, etc).
    - Unir los distintos edificios mediante un sistema de caminos y carreteras coherentes.
    - Diseñar distintos edificios y decoraciones para aumentar el valor estético del asentamiento.

- A través de las métricas mencionadas anteriormente, comparar asentamientos generados usando distintos parámetros, con el fin de hallar los valores óptimos para estos parámetros.

- Desarrollar una herramienta que cumpla los requisitos necesarios de cara a la participación en *The GDMC Competition*.

1.4 Metodología y plan de trabajo

Dadas las características del proyecto, durante su realización se ha seguido una metodología basada en la paralelización, para ello se dividen los algoritmos en tres grandes módulos: "Preprocesado", "Planificación del Asentamiento" y "Edificación del Asentamiento".

En primer lugar, se establece una funcionalidad base para las distintas partes de estos módulos, y una vez definido el flujo del programa se itera sobre los distintos algoritmos para refinar el resultado de cada uno de ellos.

Al ser tres integrantes del grupo se ha paralelizado la carga de trabajo, cada miembro se encarga de una de las tres etapas del algoritmo final. Sin embargo, todos han contribuido en cierta medida en todas las partes del proyecto.

Dividir el desarrollo de esta forma es eficiente ya que permite trabajar teniendo el mínimo número de dependencias posibles, optimizando así el tiempo de producción.

Se utiliza *git* como sistema de control de versiones para el desarrollo del proyecto, esto permite gestionar de forma eficiente y confiable el código fuente desde un repositorio privado accesible de manera online.

Ya que uno de los objetivos del proyecto es su presentación en la *GDMC*, ciertas decisiones de diseño se han tomado teniendo en cuenta las reglas de la competición.

Plan de trabajo

El diagrama de Gantt que se muestra en las Figuras 1, 2 y 3 contempla las actividades a realizar durante el desarrollo del trabajo, el objetivo de este es la planificación y gestión de las tareas para la coordinación de los miembros del grupo.

El trabajo se ha desarrollado en un periodo de 9 meses (octubre 2019 – junio 2020) por lo que se ha establecido una planificación dividida en 3 hitos:

El primer hito contempla las tareas realizas desde octubre hasta diciembre, el desarrollo se ralentizó durante las primeras semanas debido a la migración del proyecto de *MCEdit* a *Forge*.

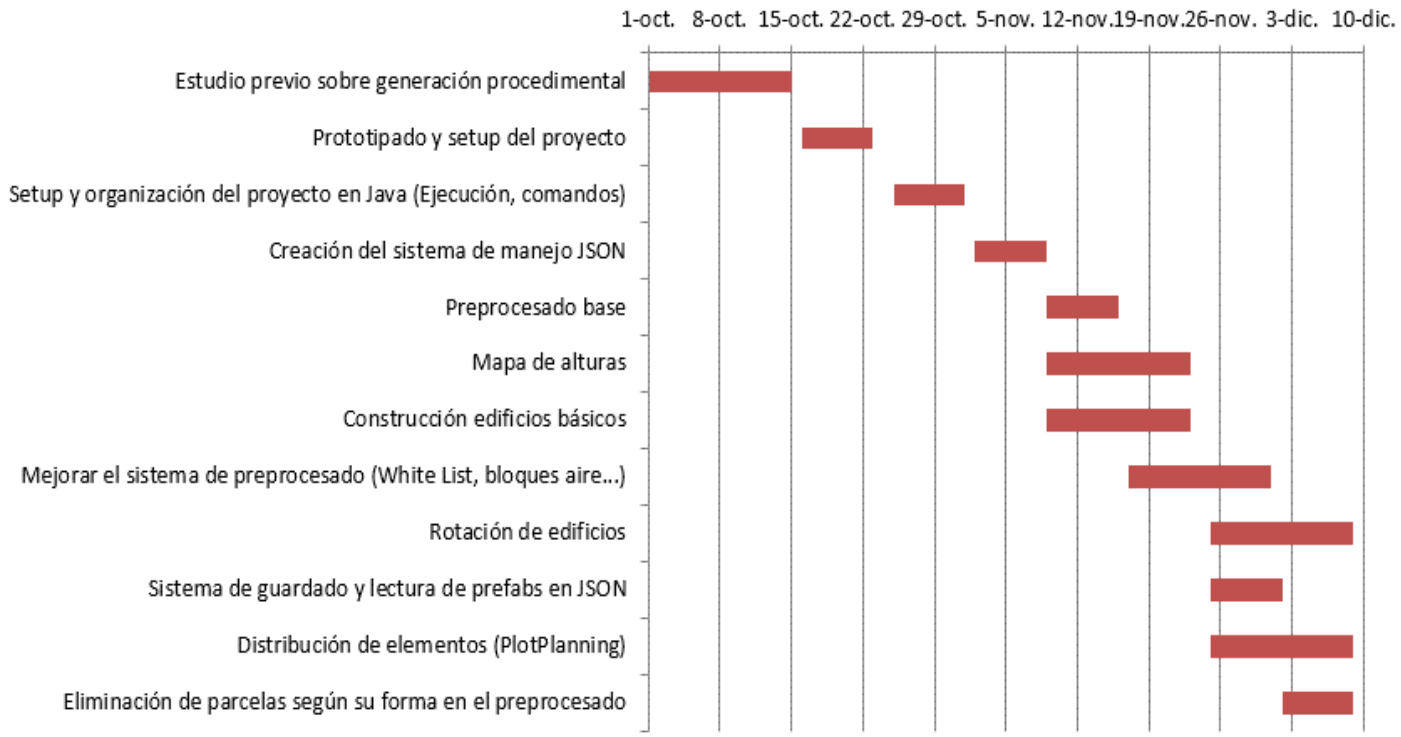

Figura 1 - Tabla de planificación del hito 1 del proyecto.

Debido al periodo de exámenes se ha decidido que el segundo hito empiece en febrero y termine en la primera semana de abril.

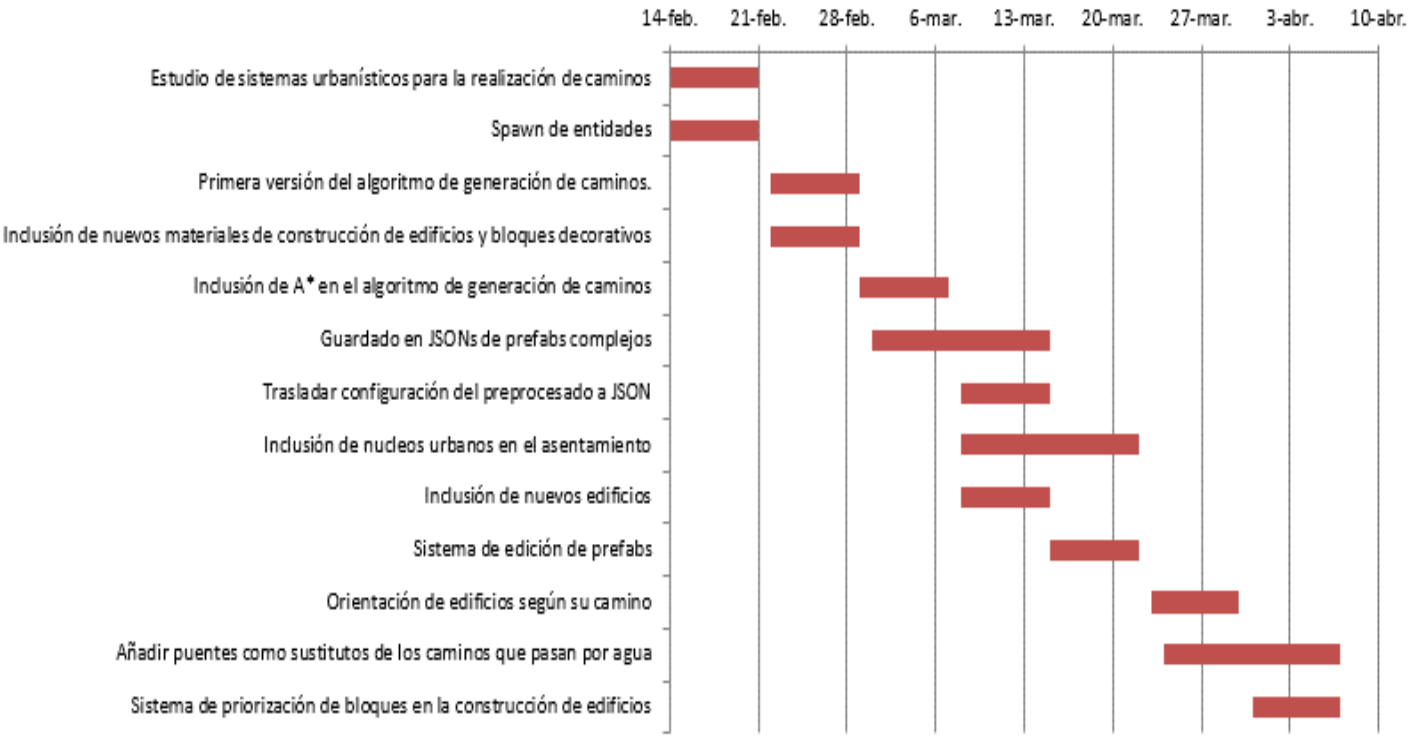

Figura 2 - Tabla de planificación del hito 2 del proyecto.

Para finalizar, el tercer hito contiene las tareas asignadas desde abril hasta la entrega del trabajo en junio.

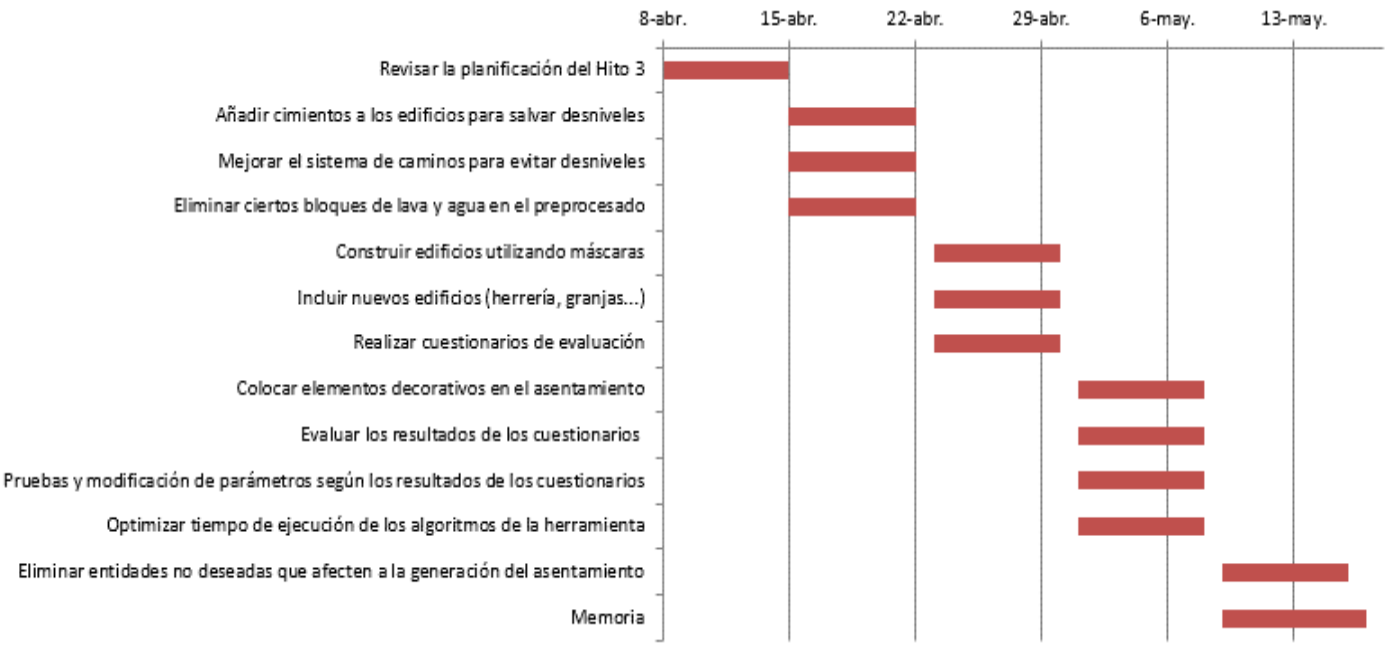

Figura 3 - Tabla de planificación del hito 3 del proyecto.

En la realización de cada tarea se tiene en cuenta la documentación de esta en la memoria.

Con el fin de concretar el alcance y objetivos del proyecto se han realizado múltiples tutorías con el director del TFG. En estas reuniones se han barajado y discutido distintas hipótesis con el propósito de alcanzar el resultado final que se buscaba.

Tras definir la hipótesis de partida se han concretado reuniones con el tutor cada 2 semanas con el fin de mostrar los avances realizados y realizar consultas sobre los problemas que pueden surgir durante el desarrollo.

Los miembros del grupo realizan reuniones con un ritmo más constante, dos veces a la semana se ponen en común los avances conseguidos, se comparten los resultados y se repasa la asignación de tareas futuras para aclarar todas las dudas que pueden surgir.

1.5 Estructura del documento

Este documento consta de 11 secciones resumidas a continuación para una presentación rápida del contenido de cada parte.

Parte 1: Introducción.
Se tratan temas como la motivación del proyecto, la hipótesis de partida, los objetivos, la metodología y el plan de trabajo.

Parte 2: Estado del Arte.
Se desarrolla en profundidad los estudios conocidos previos al desarrollo.

Parte 3: Arquitectura y plataforma.
En este apartado se explica la arquitectura utilizada en el desarrollo de la herramienta, el sistema de representación del mapa, el sistema de guardado y el uso de los comandos de *Minecraft* propios de la herramienta.

Parte 4: Preprocesado del terreno.
Descripción en detalle del algoritmo que prepara el terreno.

Parte 5: Planificación del asentamiento.
Se exponen los algoritmos utilizados para escanear el terreno y distribuir los elementos en el asentamiento de forma óptima.

Parte 6: Edificación del asentamiento.
Apartado en el que se explica el desarrollo técnico de los algoritmos encargados de los elementos de la representación gráfica, la construcción de edificios, la generación de entidades y la colocación de caminos.

Parte 7: Pruebas.
Se discuten los resultados obtenidos tras la realización de cuestionarios con el fin de evaluar distintas características de la herramienta.

Parte 8: Discusión.
En este apartado se interpretan los resultados obtenidos tras la realización del trabajo, así como sus fortalezas y limitaciones.

Parte 9: Conclusiones y trabajo futuro.
Se responde a la hipótesis planteada y se proponen distintas formas de ampliar el trabajo realizado.

Parte 10: Contribución personal.
Descripción en detalle de las aportaciones realizadas por los miembros del equipo.

Parte 11: Bibliografía
Referencias utilizadas para la realización del trabajo.

1.6 Glosario

Aquí se explica el significado de términos importantes que aparecen a lo largo del documento.

A Estrella: Algoritmo de búsqueda que se emplea para buscar el camino óptimo posible entre dos puntos.

API: Conjunto de funciones y procedimientos que se utilizan para el desarrollo de software.

Clúster: En el contexto del proyecto, se refiere a un clúster como una agrupación de parcelas o de otros clústeres.

Forge: API para programar modificaciones de *Minecraft* en Java.

GDMC: (*Generative Design in Minecraft*) Competición anual en la que distintos participantes generan algoritmos para producir asentamientos de forma procedimental en Minecraft.

IA: Siglas de "Inteligencia Artificial", es un campo de la informática destinado a simular comportamientos de la realidad.

Ítem: Objeto que aporta alguna funcionalidad o propiedad en un videojuego.

JSON: Formato de texto fácilmente interpretable para humanos y que está constituido principalmente por pares nombre/valor.

Mod: Extensión del software que permite la inclusión de nuevas características mediante la modificación del videojuego original.

NPC: Del inglés "*Non-Playable Character*", es un término utilizado en videojuegos para referirse a aquellos personajes que no controla el jugador y que generalmente responden a algún tipo de IA.

Plot: Concepto usado en la investigación para referirse a las parcelas que constituyen el asentamiento.
Como la edificación es el último eslabón de la simulación, evitamos referirnos a las unidades que forman el asentamiento como "edificios", usando el término parcela o *plot*.

Prefab: Concepto utilizado en nuestra herramienta que se refiere a objetos prefabricados que sirven como modelos que permiten hacer copias exactas instanciadas.

Sandbox: Género al que pertenecen los juegos no lineales que dan al jugador gran libertad de acción.

Vóxel: Unidad cúbica que compone un objeto tridimensional.

Vóxel Art: Estética que evoluciona del *píxel art* y utiliza vóxeles para representar entornos tridimensionales.

2. Introduction

Procedural generation is a resource increasingly used in the video game industry. *(Frade, M., de Vega, F. F., & Cotta, C., 2012).* The ability to create randomly brings variability to games and reduces the workload in almost all aspects of development such as programming, art and design. *(Shaker, N., Togelius, J., & Nelson, 2016)* These advantages have led to its use in other industries such as film, allowing the quick generation of scenarios and models.

The purpose of the study is the development of a tool that relieves this workload by generating virtual settlements in *Minecraft (Persson, M., & Bergensten, J., 2011),* taking advantage of the terrain created by the generator included in the game, and taking into account that all the elements are coherent with each other and attractive, in order to achieve as a result a map capable of competing with one generated by hand.
The environment chosen for the development of the project is *Minecraft*, one of the greatest exponents of the procedural generation in video games, being one of the best-selling games in history. *(Ames, M. G., & Burrell, J., 2017).*

In addition to being a best-seller, its terrain generation model makes it an ideal testing ground for developing procedural generation algorithms that require a complex and realistic geological base.
When generating the settlement, a set of algorithms will be used to perform the following procedures:

First, the terrain is pre-processed, these algorithms are in charge of analyzing the construction area and treating the terrain in order to create a more consistent surface for the construction algorithm.

Once the preparation of the land is done, the distribution algorithms of plots will be used. These algorithms choose the optimal places where to build each building.

Finally, once all the buildings are assigned, it is the turn of the building phase, in which the different plots are joined by paths and the different elements are built using previously designed templates.

The work has been done with a view to participating in *The GDMC Competition*, an annual competition in which different participants generate algorithms to produce settlements in a procedural way in *Minecraft*. The challenge is to create an algorithm that produces a consistent settlement on a random map. These settlements must be adaptable to the terrain, functional, and have an interesting narrative and aesthetic. *(Salge, C., Green, M.C., Canaan, R., & Togelius, J., 2018).*

2.1 Motivation of the study

The procedural generation of levels is necessary in the video game industry, since more and more developers want to devote their efforts to other sections of their projects such as narrative or development of mechanics, so this tool allows them to save time in generating maps for their experiences, prototypes or video games. (Khalifa, A., Perez-Liebana, D., Lucas, S. M., & Togelius, J., 2016)

As this is a growing industry, this type of tool is being used more and more in different areas of development, since in addition to saving resources, the opportunity to add a random component to the creation of the maps opens the door to new design possibilities and provides replay value. Among the uses of this technique is the creation of natural elements such as biomes, ecosystems and even planets and galaxies.

Specifically, this work focuses on the generation of settlements since map design is one of the most complex parts of video game development.

As for the platform on which the tool is going to be developed, the videogame Minecraft has been chosen because, in addition to being a best seller with a huge community, it has a realistic terrain generation model that simulates complex geological environments. This allows us to check the behaviour of the algorithms in a large number of situations. On the other hand, its graphic representation is based on voxels, which facilitates construction and allows for easy management of the general world by handling it as a three-dimensional matrix composed of cubes.

As mentioned above, the community is continuously growing, this favours the use of this type of tools since there are communities focused on researching the procedural creation of maps and cities for videogames, in which users share their mods and seeds of the worlds they have generated, allowing the rest to replicate them and visit curious or interesting places.

The use of this type of tool in the field of education is also noteworthy. (Ekaputra, G., Lim, C., & Eng, K. I., 2013). Through the use of Minecraft mods, it is possible to generate useful environments when teaching subjects such as biology, physics, chemistry, geology, etc. (Short, D., 2012).

2.2 Starting hypothesis

The aim of this work is to develop a tool that automatically generates settlements that are coherent, functional and visually attractive to the human eye. As creativity is a fundamental component, it is not possible to measure the factors that make a settlement meet these characteristics, so hypotheses have to be created based on subjective approaches, using metrics that simulate human behavior to improve the evaluation and perception of the final result.

For the creation of settlements, it is also interesting to add a "human" component that gives meaning to the final result, so that the villages appear natural and not artificially generated. Basing oneself on the criteria to create cities that a hypothetical civilization would use brings coherence to the structure of its elements and narrative value.

On this basis, the following hypotheses have been put forward:

- Hypothesis of the focal points: If first the focal points of the city are established and then the town is built according to the distance to them, a more faithful result to the radial urban model that we are trying to simulate is obtained.

- Hypothesis of the inclination: Scanning the land previously and prioritizing the plots with less inclination gives a more coherent result because the elements are built in areas with less unevenness.

- Noise hypothesis: Applying a Gaussian noise shift when distributing the plots gives a less squared and more "human" result.

- Hypothesis of the roads: If we join houses together and then join those sets together in a recursive way until only one set is left, whose center is the focus of the town, we obtain a branched road network that realistically resembles the radial urban model that we are trying to imitate.

- Hypothesis of the decoration: Eliminating previously all existing decoration in the world and placing it once the roads and buildings have been built, a more ordered result is obtained.

The previous hypotheses are summarized in the possibility of using algorithms that simulate historical architectural bases to generate a town with coherence and aesthetics similar to one designed by hand.

2.3 Objectives

The development of the work is marked by the fulfilment of the following objectives:

- Analyze the characteristics that make a city attractive and coherent with the aesthetics.
    - Carry out a study of the different urban development models in order to generate settlements that follow coherent urban development distributions.
    - Study the aesthetic elements that characterize human settlements in order to replicate them as faithfully as possible.

- Design metrics that value different aspects of the settlement obtained.

- Generate coherent settlements from maps previously generated by Minecraft, using algorithms that modify their elements.
    - Process the unevenness of the map to obtain a terrain where a village can be built without drastically affecting the original layout.
    - Determine the best way to distribute the different buildings, considering different parameters (inclination of the terrain, distance to points of interest, etc.)
    - Unite the different buildings through a system of coherent paths and roads.
    - Design different buildings and decorations to increase the aesthetic value of the settlement.

- Through the metrics mentioned above, compare settlements generated using different parameters, in order to find the optimal values for these parameters.

- Develop a tool that meets the necessary requirements for participation in The GDMC Competition.

2.4 Methodology and work plan

Given the characteristics of the project, a methodology based on parallelization has been followed during its execution. For this purpose, the algorithms are divided into three large modules: "Preprocessing", "Settlement Planning" and "Settlement Building".

First, a base functionality is established for the different parts of these modules, and once the program flow is defined, it is iterated over the different algorithms to refine the result of each one of them.

As there are three members of the group, the workload is parallelized, each member overseeing one of the three stages of the final algorithm. However, all of them have contributed to some extent in all parts of the project.

Dividing the development in this way is efficient since it allows working with the minimum number of dependencies possible, thus optimizing the production time.

Git is used as a version control system for the development of the project, this allows efficient and reliable management of the source code from a private repository accessible online.

Since one of the objectives of the project is its presentation in the GDMC, certain design decisions have been made taking into account the rules of the competition.

Work plan

The Gantt diagram shown in Figures 4, 5 and 6 contemplates the activities to be carried out during the development of the work, the objective of this is the planning and management of the tasks for the coordination of the group members.

The work has been developed in a period of 9 months (October 2019 - June 2020) so a planning divided in 3 milestones has been established:

The first milestone covers the tasks carried out from October to December, development slowed down during the first weeks due to the migration of the project from MCEdit to Forge.

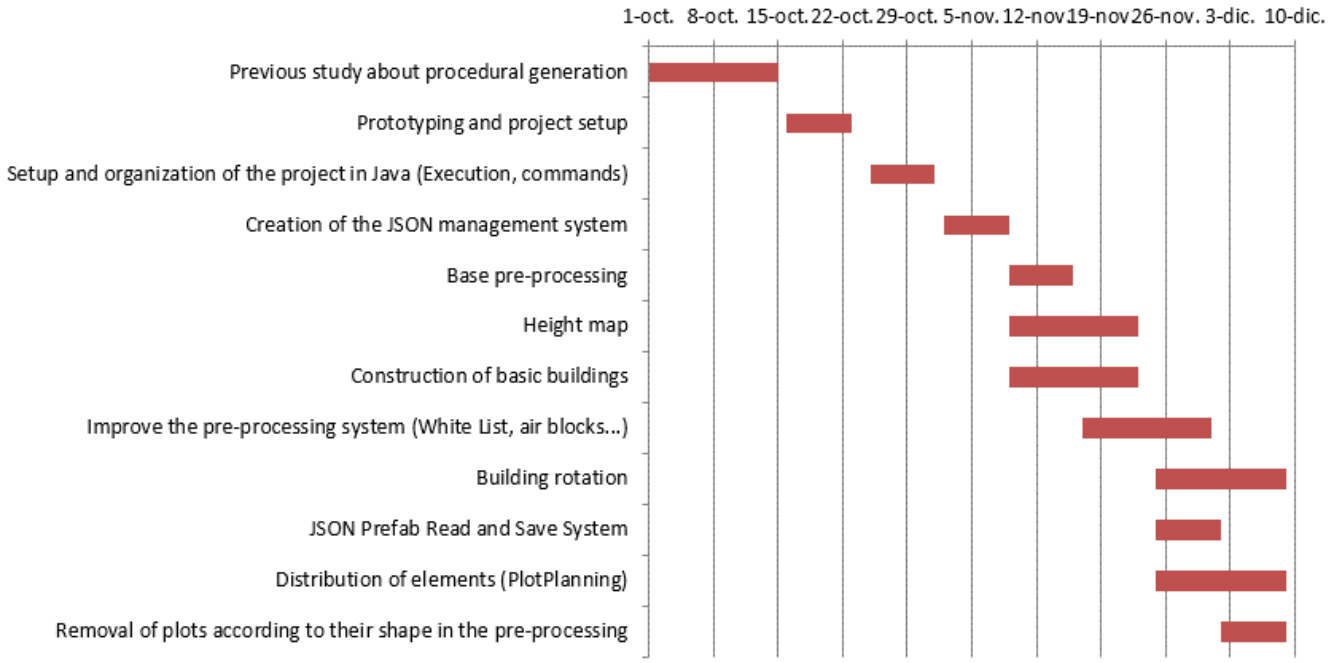

Figura 4 - Planning table for milestone 1 of the project.

Due to the examination period it has been decided that the second milestone will begin in February and end in the first week of April.

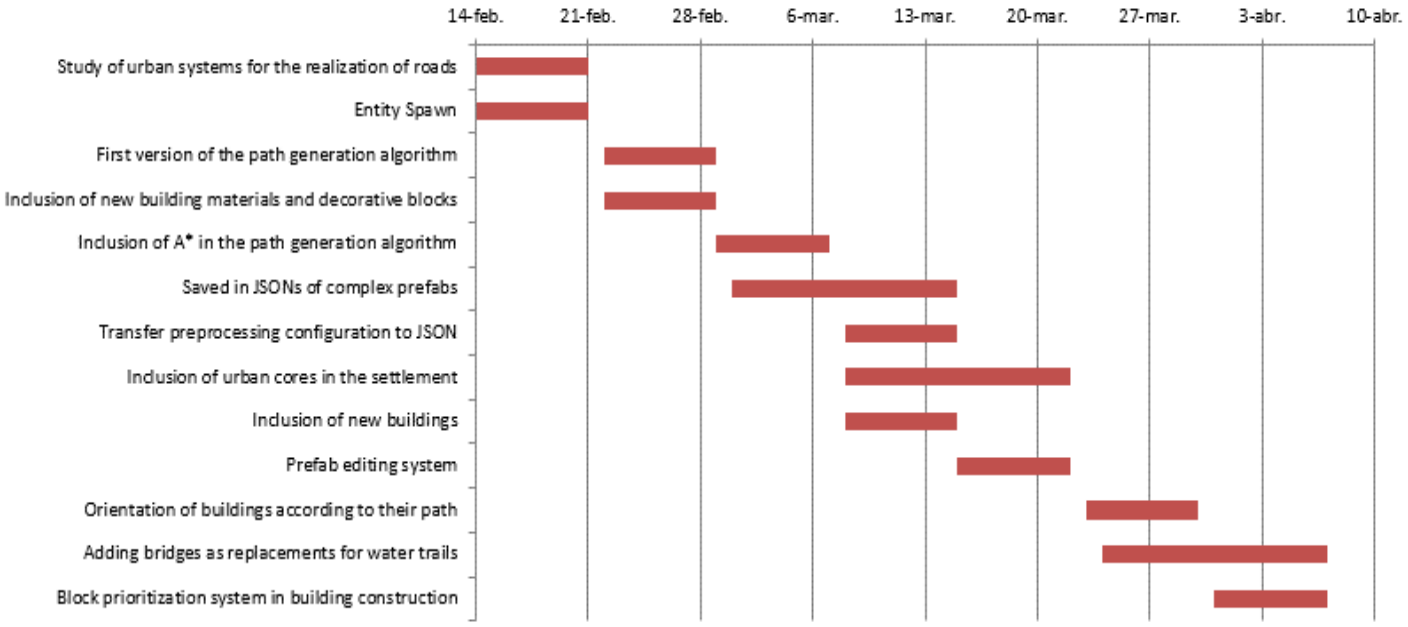

Figura 5 - Planning table for milestone 2 of the project.

Finally, the third milestone contains the tasks assigned from April until the delivery of the work in June.

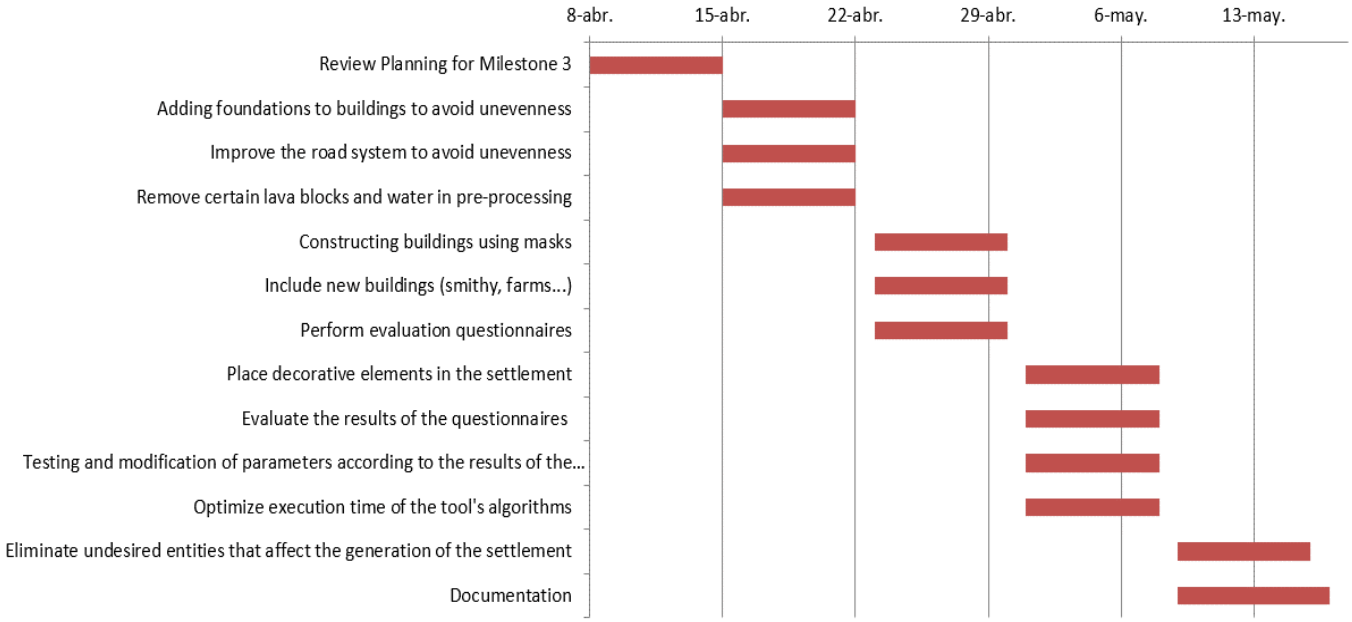

Figura 6 - Planning table for milestone 3 of the project.

When carrying out each task, the documentation of the task in the report is considered.

In order to specify the scope and objectives of the project, multiple meetings have been carried out with the director of the project. In these sessions, different hypotheses have been shuffled and discussed in order to achieve the result sought.

After defining the starting hypothesis, meetings with the director have been held every two weeks in order to show the progress made and to consult on the problems that may arise during the development.

The members of the group have meetings at a more constant pace, twice a week they share the progress made, share the results and review the assignment of future tasks to clarify any doubts that may arise.

2.5 Structure of the document

This document consists of 11 sections summarized below for a quick presentation of the content of each part.

Part 1: Introduction.
Topics such as the motivation of the project, the starting hypothesis, the objectives, the methodology and the work plan are covered.

Part 2: State of the Art.
The known studies prior to development are developed in depth.

Part 3: Architecture and platform.
This section explains the architecture used in the development of the tool, the map representation system, the saving system and the use of the tool's own Minecraft commands.

Part 4: Terrain pre-processing.
Detailed description of the algorithm that prepares the terrain.

Part 5: Planning the settlement.
The algorithms used to scan the terrain and distribute the elements in the settlement in an optimal way are presented.

Part 6: Building of the settlement.
Section that explains the technical development of the algorithms in charge of the elements of the graphic representation, the construction of buildings, the generation of entities and the placement of paths.

Part 7: Testing.
The results obtained after carrying out questionnaires are discussed in order to evaluate different characteristics of the tool.

Part 8: Discussion.
This section interprets the results obtained after carrying out the work, as well as its strengths and limitations.

Part 9: Conclusions and future work.
It responds to the hypothesis raised and proposes different ways to expand the work done.

Part 10: Personal contribution.
Detailed description of the contributions made by the team members.

Part 11: Bibliography
References used to carry out the work.

2.6 Glossary

Here we explain the meaning of important terms that appear throughout the document.

A Star: A search algorithm used to find the optimal possible path between two points.

API: Set of functions and procedures used for software development.

Cluster: In the context of the project, it refers to a cluster as a grouping of plots or other clusters.

Forge: API for programming Minecraft modifications in Java.

GDMC: (Generative Design in Minecraft) Annual competition in which different participants generate algorithms to produce procedural settlements in Minecraft.

AI: Acronym for "Artificial Intelligence", it is a field of computer science aimed at simulating behaviours of reality.

Item: Object that provides some functionality or property in a video game.

JSON: A text format that can be easily interpreted by humans and is mainly made up of name/value pairs.

Mod: Software extension that allows the inclusion of new features by modifying the original videogame.

NPC: Non-Playable Character, is a term used in videogames to refer to those characters that are not controlled by the player and generally respond to some kind of AI.

Plot: Concept used in research to refer to the plots that make up the settlement. As the building is the last link in the simulation, we avoid referring to the units that form the settlement as "buildings", using the term plot.

Prefab: Concept used in our tool that refers to prefabricated objects that serve as models that allow us to make exact instantiated copies.

Sandbox: A genre of non-linear games that give the player great freedom of action.

Voxel: Cubic unit that makes up a three-dimensional object.

Voxel Art: Aesthetics that evolves from pixel art and uses voxels to represent three-dimensional environments.

3. Estado del arte

En este apartado se describen aplicaciones que emplean técnicas similares a las utilizadas en el proyecto o que han sido relevantes para el desarrollo de las tecnologías relacionadas con la generación procedimental.

3.1 Generación Procedimental

La generación procedimental consiste en la creación de contenido de forma automática mediante la utilización de algoritmos.
Actualmente, es un campo de investigación que se encuentra en auge, ya que tiene múltiples ventajas, siendo una de ellas la inmensa cantidad de posibilidades que ofrece la aleatoriedad y que encuentra su uso en múltiples aplicaciones en el mundo de la informática. *(Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C., 2011).*

3.1.1 Elementos Pseudo-Aleatorios

Cuando se genera contenido de forma procedimental, es frecuente la búsqueda de variedad de resultados. Para ello se suele usar algún tipo de aleatoriedad que aporte estas variaciones.
Sin embargo, un generador normal es capaz de generar secuencias aleatorias muy caóticas y que carecen de coherencia entre sí. *(Vuontisjärvi, H., 2014)*

Para evitar eso, es común el uso de distintos tipos de pseudo-aleatoriedad, que, aunque mantiene la aleatoriedad buscada, lo hace de modo que los valores sean más coherentes entre sí, como se ve en la Figura 7.

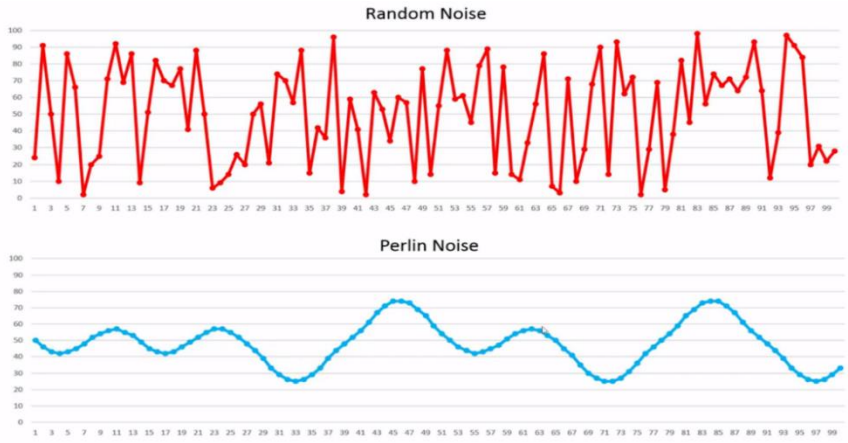

Figura 7 - Comparación entre un generador aleatorio estándar frente a uno pseudoaleatorio.

3.1.2 Ruido de Perlin

Dada la necesidad que se tiene en el desarrollo de videojuegos de encontrar algoritmos cuyo coste no sea muy elevado, el algoritmo desarrollado en 1983 por *Ken Perlin*, aporta un resultado muy natural mientras mantiene una complejidad baja*. (Amato, A., 2017)*

El *Ruido de Perlin* se suele tratar como una función que para una coordenada devuelve un valor entre 0 y 1.
Visualmente esto se puede representar en una escala de grises como la que se puede apreciar en la Figura 8, donde el 0 es negro y el 1 blanco.

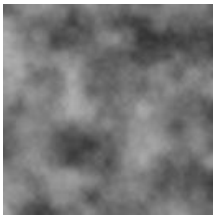

Figura 8 - Textura generada procedimentalmente mediante Perlin.

Para establecer los valores en el espacio se realizan tres pasos:

- Definir la matriz y los vectores de gradiente:
  Se establecen en los distintos vértices de la matriz diversos vectores de gradiente generados pseudoaleatoriamente como se ve en la Figura 9.
  Esta pseudoaleatoriedad permite que para la misma entrada siempre se genere la misma salida, siempre y cuando no se cambie el algoritmo generador.

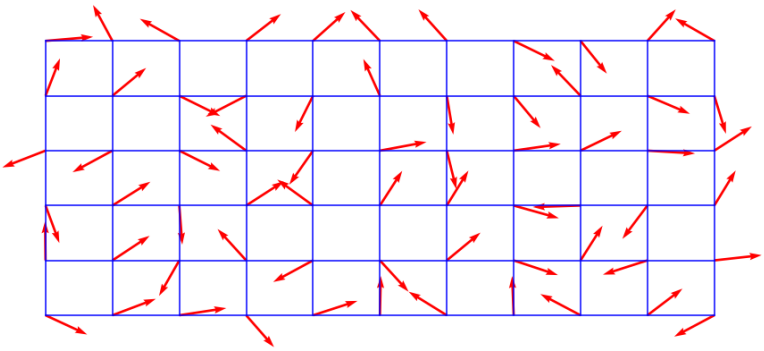

Figura 9 - Matriz de vectores generados aleatoriamente. (https://en.wikipedia.org/wiki/File:PerlinNoiseGradientGrid.png)

- Producto escalar:
  A continuación, se computa cada vector de gradiente en relación con la coordenada introducida.
  Para cada nodo se realizan dos operaciones:
  - Calcular el vector distancia hasta el punto definido en la entrada.
  - Realizar el producto escalar con el vector de distancia previamente obtenido. En la Figura 10 se muestra gráficamente el resultado de la operación.

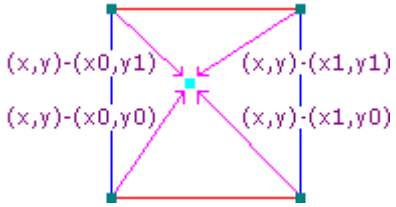

Figura 10 – Visualización de el computo realizado a cada vértice con relación a la coordenada introducida.

Al realizar este proceso para cada nodo se obtiene el resultado mostrado en la Figura 11.

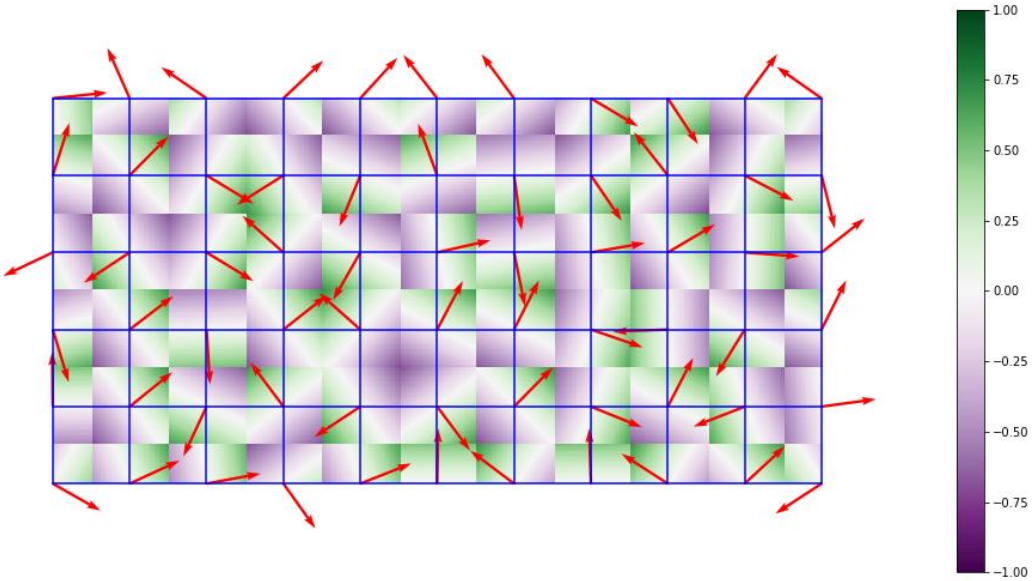

Figura 11 – Resultado de aplicar la formula anterior a todos los vértices

(https://en.wikipedia.org/wiki/File:PerlinNoiseDotProducts.png)

- Suavizado:
  En el resultado anterior se puede observar la falta de cohesión entre los diversos nodos. Para lograr un resultado más suave se realiza una interpolación entre los valores respectiva a los nodos utilizando una función similar a la mostrada en la Figura 12, obteniendo así valores que varían de forma más gradual como se puede ver en la Figura 13.

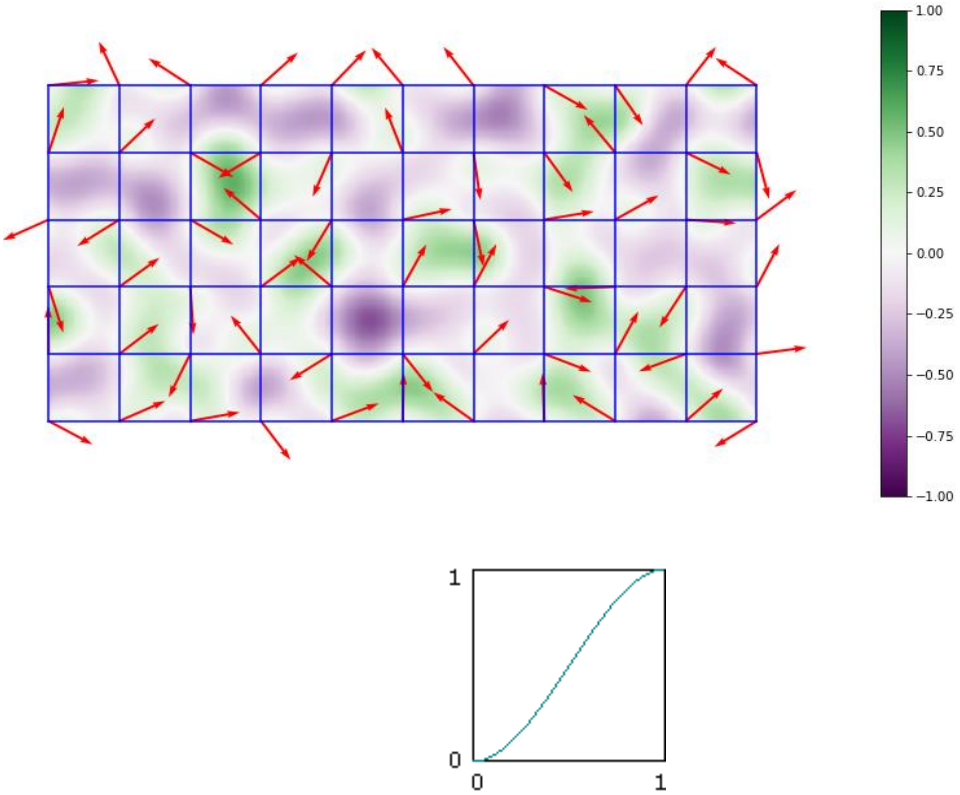

Figura 12 - Ejemplo de una función utilizada para suavizar los gradientes.

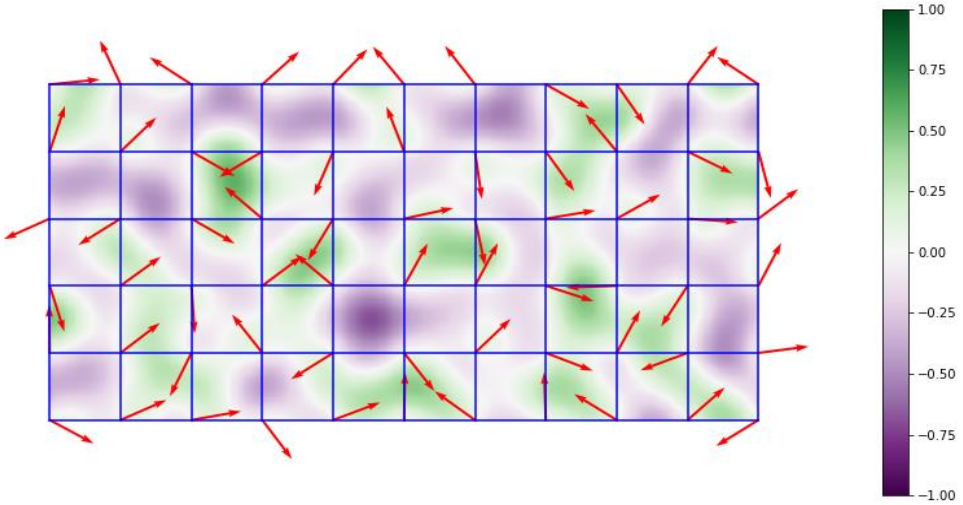

Figura 13 - Resultado final tras aplicar la interpolación.

(https://en.wikipedia.org/wiki/File:PerlinNoiseInterpolated.png)

Gracias a estas características, el algoritmo permite simular una naturalidad y coherencia que puede ser usada para la creación de diversos contenidos. *(Amato, A., 2017)*

Destaca en el campo de la generación de terreno, ya que permite simular la irregularidad del terreno mientras garantiza la coherencia de este.
Se puede agrupar el terreno natural en distintos niveles de detalle:

- Topografía general: Elementos a gran escala como es la elevación general del terreno (montañas y valles).

- Topografía local: Distintos elementos de una montaña, como las distintas colinas o depresiones.

- Detalle a gran escala: Elementos más concretos, pero con cierta repercusión en el terreno, como grandes rocas.

- Detalles pequeños: Elementos más pequeños que forman el suelo, como pequeñas rocas y variaciones del suelo.

Aplicando la función de ruido con distintos valores de amplitud y frecuencia se pueden obtener niveles de detalle como los que se pueden ver en la Figura 14. *(Li, H., Yang, H., & Zhao, 2017)*

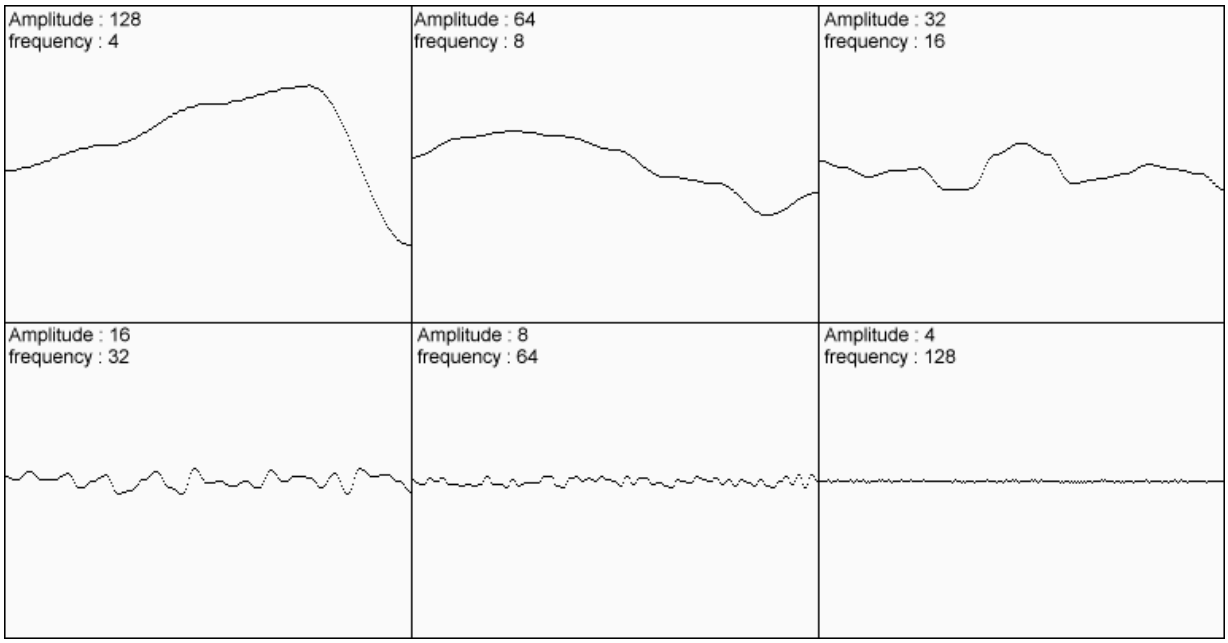

Figura 14 – Visualización de variar la amplitud y la frecuencia en la fórmula.

Juntando todos los niveles se puede obtener un resultado como en el mostrado en la Figura 15.

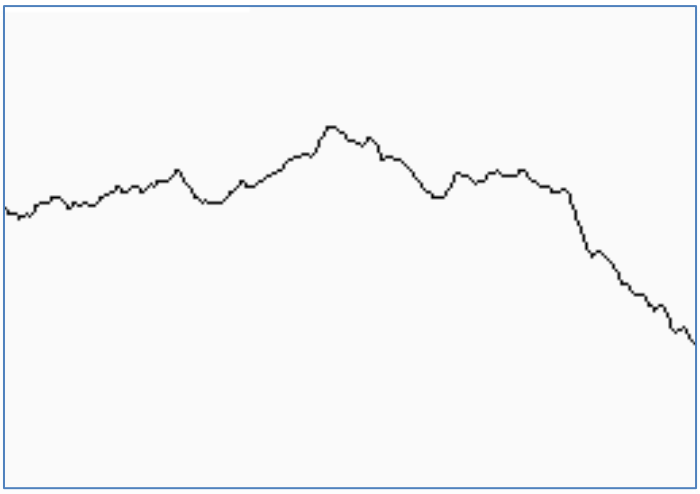

Figura 15 - Resultado de juntar las distintas octavas generadas anteriormente.

3.1.3 Generación procedimental en el cine

La generación procedimental se utiliza a menudo en el cine con el fin de crear efectos visuales sin necesidad de artistas, se pueden destacar la creación de espacios virtuales, la simulación de sistemas de partículas o la generación de texturas a partir de funciones matemáticas. *(Kerlow, I. V., 2009)*

Otra posible aplicación es la generación de gran número de objetos similares a partir de un modelo, de esta forma el artista puede replicar el objeto de forma imperfecta aplicando ligeras variaciones al original dando la sensación de que no es una copia.

Uno de los máximos exponentes de creación procedimental en el cine es *Massive*, un software de animación que utiliza inteligencia artificial para generar diversos efectos visuales *(Thalmann, D., Hery, C., Lippman, S., Ono, H., Regelous, S., & Sutton, D., 2004). Massive* se ha utilizado en películas de gran escala para simular enormes grupos de entidades independientes, destacando entre ellas la trilogía del *Señor de los Anillos* para simular ejércitos multitudinarios *(Aitken, M., Butler, G., Lemmon, D., Saindon, E., Peters, D., & Williams, G., 2004)* o la película *Ratatouille* donde aparecían grandes conjuntos de ratas simultáneamente. *(Ryu, D., & Kanyuk, P., 2007)*

3.1.4 Generación procedimental de ciudades virtuales

Ciertas industrias como la del videojuego o el cine necesitan generar entornos cada vez más amplios y detallados, siendo uno de los mayores retos la creación de paisajes urbanos complejos.

Puesto que diseñar paisajes urbanos es complicado debido a la gran complejidad y a la cantidad de elementos que los conforman, se han creado aplicaciones que, mediante el uso de técnicas de generación procedimental, ayudan a construir estos entornos reduciendo el tiempo y coste de producción sustancialmente. *(Kelly, G., & McCabe, H., 2006)*

Una de ellas es *Citygen* una herramienta cuya finalidad es automatizar la creación de paisajes urbanos, entre sus ventajas se encuentra la capacidad de modificar los parámetros de generación del algoritmo fácilmente por medio de una interfaz visual, dando además la posibilidad de ver estos cambios en tiempo real. *(Kelly, G., & McCabe, H., 2007)*

3.1.5 Generación procedimental en los Videojuegos

La generación procedimental se empezó a emplear casi desde los orígenes de la industria del videojuego. Motivado por la falta de memoria, se buscaron maneras de generar los niveles y texturas sin tener que almacenarlos, naciendo así los primeros mapas generados procedimentalmente.

Esta técnica se encuentra presente en la mayoría de los ámbitos del desarrollo de videojuegos como son la generación de narrativa *(Grey, J., & Bryson, J. J., 2011),* puzzles *(Fernández-Vara, C., & Thomson, A., 2012)*, niveles *(Compton, K., & Mateas, M., 2006),* música *(Collins, K., 2009), etc.*

Aunque ya se había utilizado anteriormente en juegos, algunos de los títulos que más influyeron y que marcaron un punto de inflexión fueron *Rogue* (1980) y *Dwarf Fortress* (1984).

Rogue

Rogue es un juego de exploración de mazmorras que se publicó en 1980 desarrollado por *Michael Toy* y *Glenn Wichman*, inspirados en juegos de fantasía basados en texto como *Star Trek* (1971). Unificaron mazmorras generadas procedimentalmente con ítems representativos de los juegos *RPG "Dragones y Mazmorras",* movimiento por turnos y el concepto de muerte permanente. Estas características sentarían las bases del género *Roguelike*, que sigue siendo popular incluso hoy en día. Las mazmorras constan de una sucesión de niveles que se

crean procedimentalmente, cada nivel tiene salas rectangulares que se conectan entre sí mediante pasillos como se puede ver en la Figura 16. Los objetos que aparecían, así como las escaleras para pasar al siguiente también eran aleatorios. (*Garda, M. B., 2013).*

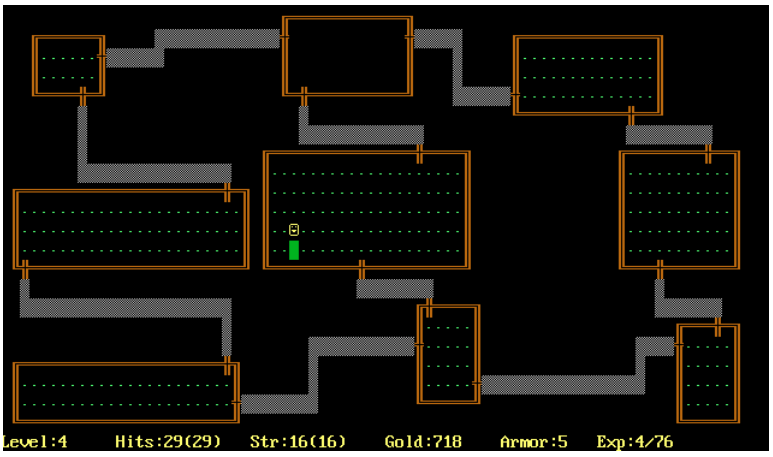

Figura 16 - Rogue (1980) Ejemplo de la disposición de un nivel.

(https://es.wikipedia.org/wiki/Rogue#/media/Archivo:Rogue_Screen_Shot_CAR.PNG)

Dwarf Fortress

Al hablar de generación procedimental no se puede olvidar *Dwarf Fortress*. *(Adams and Adams; Hall, 2014)*
Este videojuego permite al jugador liderar una colonia de enanos en la construcción de una fortaleza, si bien este planteamiento es sencillo el juego brilla en el desarrollo de sus múltiples sistemas y la generación del mundo que presenta.

La generación del mapa de *Dwarf Fortress* es procedimental y se divide en distintas fases, en primer lugar, se genera el terreno mediante el uso de distintos algoritmos que seleccionan los emplazamientos para las elevaciones y los puntos bajos del mismo. Tras esto a cada localización del terreno se le asignan las precipitaciones anuales y en la siguiente fase se selecciona la disposición de los distintos minerales, la temperatura y el clima. La fusión de todos estos apartados da lugar a la formación de los distintos biomas. *(Adams, T., 2015)*

Una vez se han generado los biomas se simula sobre ellos la erosión en el paso del tiempo y se sitúan los ríos, valles, océanos… Posteriormente un algoritmo define la salinidad de las zonas creando así islas, ciénagas, etc.

A la hora de generar picos nevados, bosques tropicales y desiertos se utiliza un algoritmo que genera sombras orográficas, este simula las corrientes de aire húmedo que remontan obstáculos topográficos (por ejemplo, las montañas del terreno) y descargan su humedad en forma de lluvia en el lado de barlovento, tras esto el aire seco desciende por sotavento generando terrenos secos.

Con el fin de conseguir terrenos coherentes se establecen unos criterios que cada fase debe cumplir, si no se cumplen se vuelve a generar la fase hasta dar con un resultado válido como el que aparece en la Figura 17.

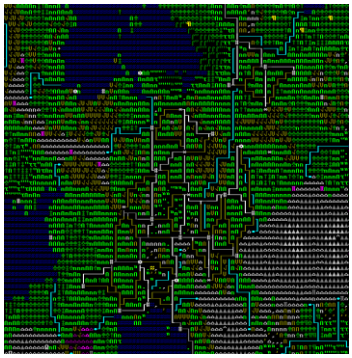

Figura 17 - Mapa generado procedimentalmente en Dwarf Fortress.

Uno de los puntos fuertes de *Dwarf Fortress* es su narrativa procedimental, tras modelar el terreno se simula el paso de cientos de años en él creando así las distintas culturas que poblarán el mundo, desarrollando ciudades, generando dinastías y dotando de un trasfondo a cada localización que aparece en el juego. *(Ryan, J. O., Mateas, M., & Wardrip-Fruin, N., 2015).*

El juego cuenta incluso con un sistema de energía, a cada región del mapa generado se le asigna un nivel de energía que va desde la positiva a la negativa, este parámetro se utiliza para generar de forma procedimental los nombres de las regiones.

3.2 Minecraft

El videojuego sobre el que se desarrolla el proyecto es *Minecraft*, un juego de construcción de tipo *sandbox* basado en bloques.

Se trabaja sobre *Minecraft* porque permite generar mapas aleatorios con distintos biomas y tipos de terreno, gracias a ello se puede probar el algoritmo en una gran variedad de situaciones y comprobar cómo se adapta a ellas. Además, aporta toda la parte gráfica del proyecto.

3.2.1 Historia de Minecraft

Minecraft es un videojuego de construcción creado por *Markus Persson* (más conocido como *"Notch"*), es uno de los videojuegos más vendidos de la historia y

entre sus influencias se encuentran *Dwarf Fortress* y *Dungeon Keeper*. *(Duncan, S. C., 2011).*

En 2009 *Notch* muestra por primera vez su proyecto, un clon de *Infiniminer* que sería la primera versión de *Minecraft*.

Una versión primeriza de *Minecraft* fue publicada por *Notch* en *TIG* un foro donde desarrolladores independientes comparten sus proyectos para recibir sugerencias y opiniones sobre ellos.

Esta versión tuvo un gran apoyo y *Minecraft* creció gracias a las recomendaciones de los primeros jugadores, con el tiempo se implementaron nuevas características al juego, llegaron el agua y la lava, aparecieron nuevos tipos de bloques, modo multijugador, modo supervivencia…

Gracias al gran impacto que el juego tuvo en la comunidad y a su gran cantidad de jugadores *Notch* fundó la empresa *Mojang* y comenzó la preventa de *Minecraft*.

Más tarde llegó la versión *Indev* que incluía herramientas, mesas para crear multitud de objetos, cofres, diamantes, ciclo de día y noche, armaduras y sistema de cosechas entre otras muchas mejoras. Actualizaciones posteriores añadieron a *Minecraft* nuevas funciones destacando entre ellas la inclusión de mundos infinitos y el redstone que permitió la creación de artilugios electrónicos en *Minecraft*.

En 2014 *Microsoft* adquirió *Mojang* y como consecuencia *Minecraft*, a fecha de la redacción de esta memoria la empresa sigue expandiendo el contenido del juego mediante actualizaciones.

3.2.2 Generación del terreno de Minecraft

Superficie

Aunque el algoritmo y las fórmulas exactas usadas para la generación del terreno de *Minecraft* son desconocidas, el creador, ha hablado a menudo de los procesos empleados para la generación.

En una etapa temprana del desarrollo, se empleaban métodos ya conocidos para generar el terreno como eran los mapas de alturas generados mediante ruido 2D de *Perlin*, similar al método de mapas de altura en una malla 3D, como se aprecia en la Figura 18. *(Olsen, J., 2004).*

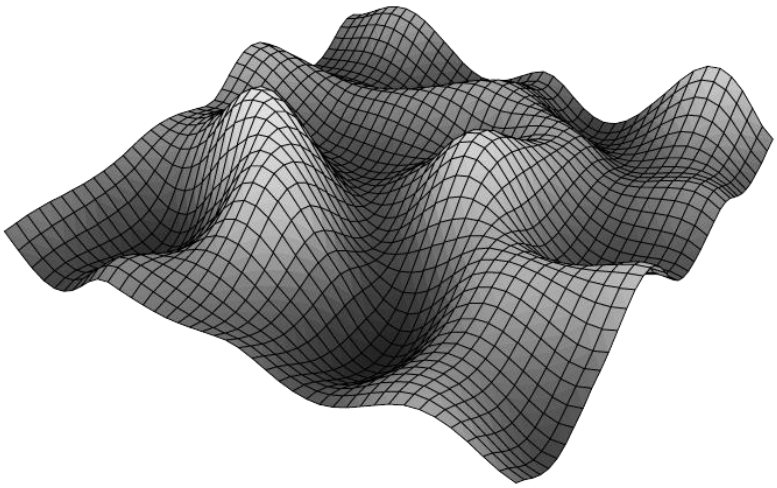

Figura 18 – Resultado de aplicar una matriz de Perlin 2D (mapa de altura) en una malla 3D.

Mediante estos procesos se obtienen mapas consistentes de una manera rápida y sin ocupar mucha memoria, pero carecía de impacto, ya que el algoritmo solo era capaz de generar superficies con poca profundidad, donde no se pueden dar elevaciones y hendiduras con salientes u otras formas naturales *(Parberry, I., 2014),* esto se puede apreciar en la Figura 19.

Con el fin de generar mapas más interesantes, se sustituyó el mapa de alturas 2D por uno similar, pero usando ruido 3D. De este modo el algoritmo ya no indicaba la altura de cierta coordenada en el plano, si no que para cada bloque del espacio se generaba una "densidad", mediante la que se decidía si el bloque fuese aire o terreno.

Gracias a este cambio, el terreno empezó a mostrar planicies, montañas, acantilados y muchas otras formas más interesantes y complejas como las mostradas en la Figura 20.

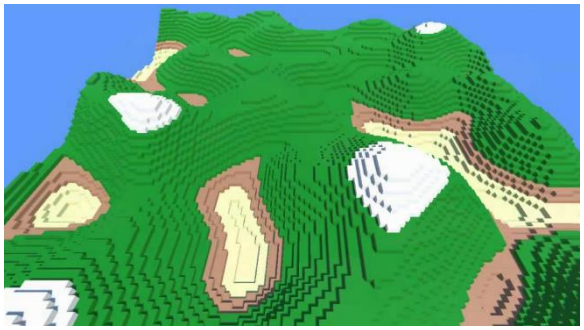

Figura 19 - Ejemplo de un mapa generado de la primera forma.

Figura 20 - Ejemplo de un mapa generado de la segunda manera.

Cuevas

Una vez se ha generado el terreno principal y la superficie, se generan las cuevas basándose en el algoritmo conocido como *"Gusanos de Perlin"* para simular el avance de un gusano por el espacio, donde las direcciones a las que se dirige vienen dadas por un ruido 2D con el fin de que el movimiento sea suave y coherente. *(Marttinen, A, 2017)*, (Bevins, J. Libnoise Glossary. *Libnoise*.).

El algoritmo empleado funciona del siguiente modo:

Primero se elige un segmento del ruido que va a ser usado para elegir las direcciones, como se observa en la Figura 21

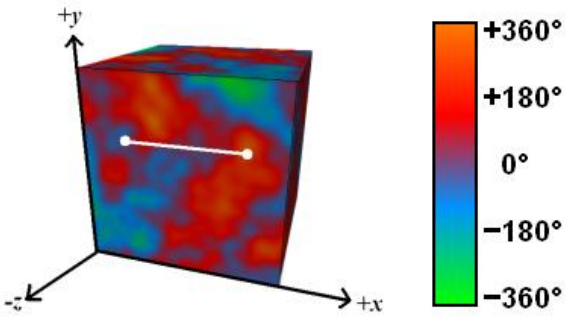

Figura 21 – Selección de un segmento de Perlin

(http://libnoise.sourceforge.net/examples/worms/images/cube.png)

Una vez escogido el segmento, este se fragmenta en valores discretos en función del número de desplazamientos que se desean. La Figura 22 y la Figura 23 representan dicha fragmentación.

Figura 22 – Segmento generado

(http://libnoise.sourceforge.net/examples/worms/images/noisebar.png)

Figura 23 – Discretización del segmento

(http://libnoise.sourceforge.net/examples/worms/images/noisebar1.png)

Una vez determinados los ángulos de desplazamiento, se vacía un área a lo largo del recorrido establecido, disminuyendo el tamaño de esta área conforme se aleja del inicio. En la Figura 24 y la Figura 25 se representa gráficamente el desarrollo del recorrido del algoritmo.

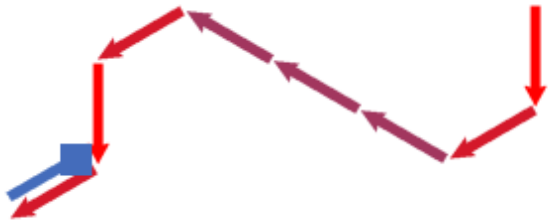

Figura 24 – Visualización de los ángulos establecidos.

(http://libnoise.sourceforge.net/examples/worms/images/worm1.png)

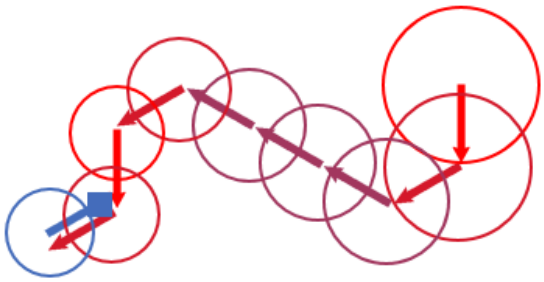

Figura 25 – Visualización de la disminución del área durante el recorrido.

Aplicando este método desde diversos puntos de la superficie, se obtiene un intrincado sistema de cuevas similar al de las Figuras 23 y 24.

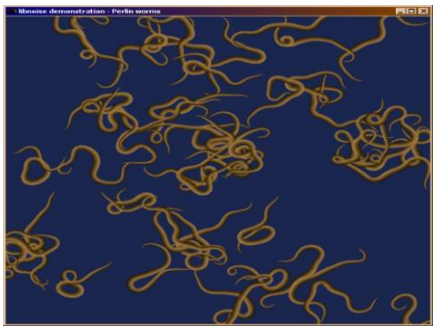

Figura 26 - Visualización del algoritmo en un entorno 3D generado por la librería de ruido "LibNoise" (Bevins, J. Libnoise Glossary. Libnoise)

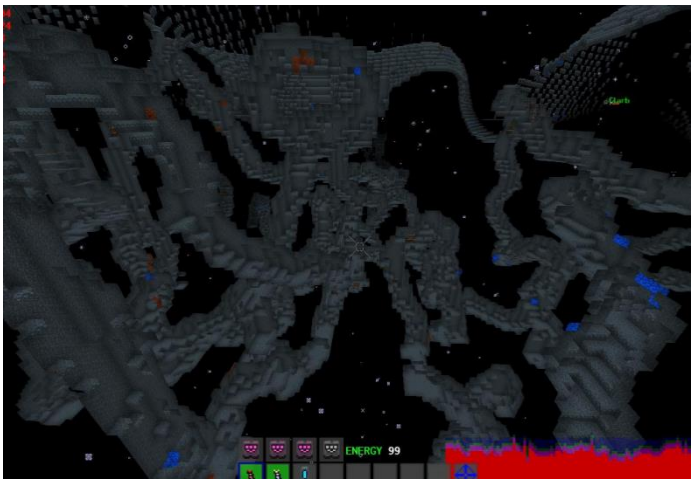

Figura 27 - Resultado del algoritmo en Minecraft.

3.2.3 Generación de asentamientos en Minecraft

Con las distintas actualizaciones el juego base ha empleado distintos algoritmos para la generar los asentamientos.

Antes del parche 1.14 los asentamientos generados presentan un aspecto similar a los de la Figura 28, los pasos para colocar este tipo de pueblos eran los siguientes:

- Busca un punto en el mapa apto (en la superficie de la tierra).
- Coloca una fuente en el centro del pueblo.
- Dibuja caminos que se extienden hacia las cuatro direcciones tomando como origen el centro.
- Crea edificios de una lista de prefabricados a lo largo de los caminos, si alguno se encuentra sobre bloques no aptos (agua, lava, aire, etc…) crea debajo de estos una base de piedras.
- Genera los aldeanos que conforman el pueblo.

A partir del parche 1.14, introducen una nueva forma de crear los pueblos, enfocada a que los jugadores pudieran crear asentamientos con estructuras propias. Se tiene un conjunto de plantillas con los edificios a construir y unos bloques especiales denominados *"jigsaw"* que determinan los puntos de acceso y que tienen la función de unir las distintas estructuras.

Se coloca la estructura central, y para cada uno de sus puntos de acceso se une otra estructura que quepa y que tenga un *"bloque jigsaw"* que encaje. De esta

forma el algoritmo va colocando los edificios donde encajan como si fueran piezas de un puzzle.

Figura 28 - Asentamiento generado automáticamente por Minecraft en la versión 1.12.

3.3 Diseño Generativo en Minecraft (GDMC)

The GDMC Competition ha servido para establecer unas bases sobre las que trabajar ya que se han seguido sus criterios de calificación a la hora de tomar ciertas decisiones de diseño, además han proporcionado la tecnología que se ha utilizado para desarrollar el proyecto.

Se trata de un concurso anual en la que distintos participantes generan algoritmos para producir asentamientos de forma procedimental en *Minecraft*. En la fecha en la que se ha realizado este proyecto se está celebrando la tercera entrega del concurso. El reto consiste en crear un algoritmo que produzca un asentamiento en *Minecraft* en un mapa provisto por el jurado y desconocido por los participantes. Los asentamientos generados deben ser funcionales, adaptarse bien al terreno del mapa y tener una narrativa interesante. *(Green, M. C., Salge, C., & Togelius, J., 2019)*
La narrativa es evaluada en profundidad en otro de los retos de *The GDMC Competition*, llamado *Chonicle Challenge*, este consiste en la generación de narrativa basada en la historia del asentamiento. *(Salge, C., Guckelsberger, C., Green, M. C., Canaan, R., & Togelius, J., 2019).*

En su página pueden encontrarse ejemplos y tutoriales creados por la comunidad con el conocimiento de años anteriores, entre ellos destacan los tutoriales sobre *MCEdit* y ejemplos en repositorios de *GitHub*.

Para participar se permiten tres formas de entrega de los proyectos: usando *MCEdit*, un mod hecho con *Forge* o un archivo *.jar* que reciba como entrada un archivo *MCEdit*.
Cada algoritmo se ejecutará en 3 mapas distintos con un tamaño de 256 x 256, aunque puede oscilar entre tamaños de 200 y 300. El resultado de cada algoritmo se evalúa por un jurado de personas que califican 4 puntos clave: Adaptabilidad, funcionalidad, narrativa y estética.

La adaptabilidad tiene en cuenta que el asentamiento encaje con el terreno y utilice materiales acordes con su bioma.

La funcionalidad mide la utilidad del asentamiento creado, si está protegido ante peligros externos, si provee de recursos a los aldeanos, si da facilidad para moverse…

La narrativa es si el asentamiento evoca alguna historia interesante, si se pueden conocer aspectos de la cultura que la rodea, si se deja clara la función del asentamiento…

La estética mide que se vea bien, que haya suficiente variedad entre los edificios, que sea realista…

La puntuación final es la media de estos 4 aspectos, que se califican entre 0 y 10, en la Figura 29 se puede ver el resultado del algoritmo ganador de la competición 2019.

Figura 29. Asentamiento generado por Filip Skwarski, ganador de la GDMC 2019.

4. Arquitectura y plataforma

El objetivo del proyecto es generar una serie de algoritmos que al ser ejecutados modifiquen un mapa de *Minecraft,* creando en él un asentamiento coherente aprovechando los recursos del terreno.

Para encontrar la herramienta adecuada sobre la que desarrollar el proyecto se ha realizado una investigación sobre proyectos similares llevados a cabo con éxito.

En primer lugar, se ha realizado un prototipo utilizando filtros en *MCEdit* programados en *Python*, tras comprobar el alcance del proyecto se ha decidido desechar este prototipo, pues se considera que *Java* es más adecuado para un trabajo de estas características, por lo que se ha creado una versión de este en *Java* utilizando para ello *Minecraft Forge*. (Una *API* que se encarga de gestionar mods de *Minecraft*).

4.1 Tecnología utilizada

Dado que el proyecto se ha realizado por varias personas se ha priorizado el uso de programas que permiten trabajar en él de forma simultánea.

El proyecto se ha desarrollado utilizando el lenguaje de programación *Java* en el *IDE Eclipse*, con el fin de asegurar y agilizar el desarrollo de la herramienta se ha recurrido a *git* como sistema de control de versiones.

En la fase de prototipado se ha utilizado *MCEdit*, un programa que permite editar mapas de *Minecraft* mediante la aplicación de filtros programados en Python.

Ya que la herramienta es un mod de *Minecraft*, durante la fase de desarrollo se ha utilizado para su implementación *Forge*, una aplicación que permite gestionar *mods*. *(Gupta, A. y Gupta, A., 2015. Minecraft Modding with Forge: A Family-Friendly Guide to Building Fun Mods in Java.)*

Los cuestionarios de evaluación que se han emitido para obtener retroalimentación se han hecho con *Formularios de Google*, puesto que es una herramienta que permite obtener datos, crear encuestas y distribuirlas de manera muy sencilla.

En cuanto a la realización de la memoria, se ha utilizado *Microsoft Word* en la plataforma *Office 365* puesto que permite a todos los usuarios modificar el documento en tiempo real.

4.2 Arquitectura secuencial de generación de asentamientos en Minecraft

Los algoritmos generados se dividen en tres bloques: "Preprocesado", "Planificación del Asentamiento" y "Edificación del Asentamiento".

En primer lugar, se ejecutan los algoritmos de preprocesado que se encargan de preparar el terreno para que las siguientes etapas puedan contar con más espacio útil a la hora de plantear las parcelas y construir el asentamiento.

Una vez el terreno ha sido procesado se da paso a los algoritmos de generación del asentamiento, estos se encargan de decidir en qué zonas se colocan las distintas construcciones. Tras esto se establece y construye la red de caminos que conectan los distintos elementos del asentamiento.

Por último, actúan los algoritmos de construcción que edifican sobre las parcelas calculadas en el paso anterior. Las casas y demás elementos que se construyen son diseñadas previamente y cargadas mediante la lectura de archivos.

4.3 Representación del mapa

Durante el desarrollo surgió la necesidad de representar el mapa del mundo como una matriz 2D. Al principio se creaban nuevas matrices conforme surgía la necesidad de un nuevo tipo de información, pero se decidió aunar los distintos tipos en un contenedor que llevara toda la información.

4.3.1 Altura

En un primer momento, se escanea el terreno por columnas guardando la altura del primer bloque distinto de aire. Esto genera algunos problemas con las zonas de agua.

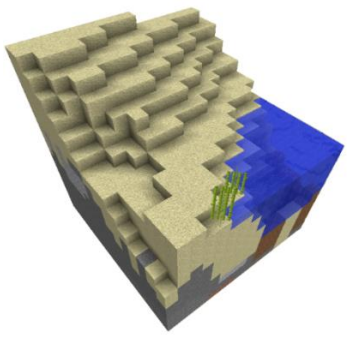

Figura 30 – Representación real del terreno.

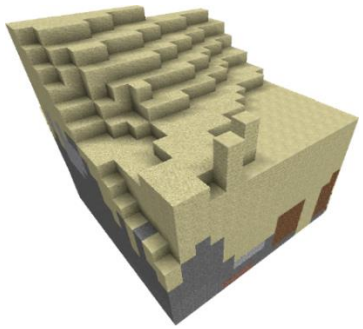

Figura 31 – Interpretación del terreno por el algoritmo.

Se puede observar que la zona de agua Figura 30 se interpreta como una zona plana Figura 31, lo que provoca que el algoritmo sea propenso a construir casas en zonas acuáticas.

4.3.2 Ocupación

Lleva la información sobre la ocupación de parcelas en el mapa. Puede ser de tres posibles tipos, excluyentes entre sí, explicados a continuación, y representados en la Figura 32, Figura 33 y Figura 34:

- **Libre**: Indica que el bloque está libre para construir.
- **Ocupado**: Indica que el bloque forma parte de una parcela
- **Espaciamiento**: Indica que el bloque no forma parte de una parcela, pero tiene una cerca, por lo que no se pueden construir parcelas ahí.

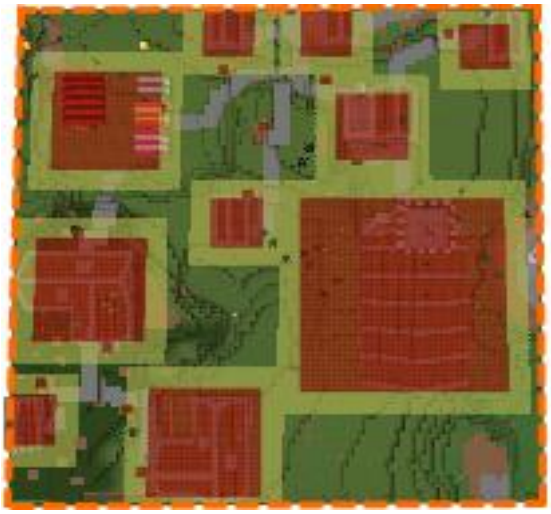

Figura 32 – Interpretación del mapa sobre el asentamiento real.

Figura 33– Asentamiento generado.

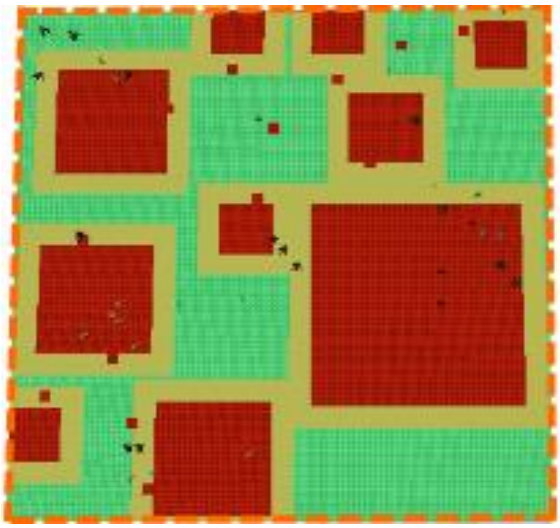

Figura 34 - Interpretación del mapa donde el rojo significa "Ocupado", el amarillo "Espaciamiento" y el verde "Libre".

4.3.3 Propiedades

Con el fin de contemplar las distintas propiedades que puede tener un objeto, se decide que cada posición tiene un vector con las propiedades que posea. Estas propiedades se pueden añadir y quitar en cualquier momento, lo que permite que los distintos algoritmos lean y escriban la información necesaria en ellos. Las propiedades contempladas son las siguientes:

- **Agua:** Indica que la posición contiene agua, como se puede ver en la Figura 35 y en la Figura 36. Esto es útil para evitar construir edificios en medio del agua. Además, permite la identificación de regiones acuáticas que permiten la edificación de puentes.

Figura 35 – Asentamiento generado.

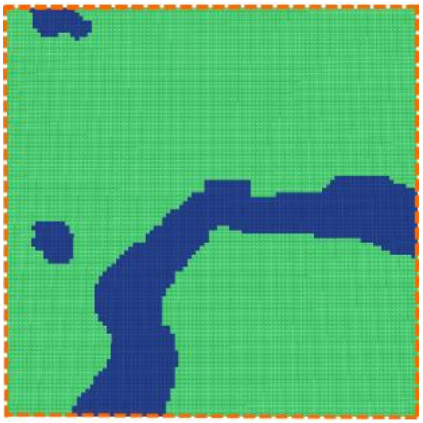

Figura 36 - Representación en el mapa, donde las zonas acuáticas se representan en azul.

- **Camino:** Una vez que se ha decidido el camino a seguir de un punto a otro, se marcan las casillas que lo forman, el resultado final se puede apreciar en la Figura 37 y en la Figura 38.

Figura 37 – Asentamiento Generado.

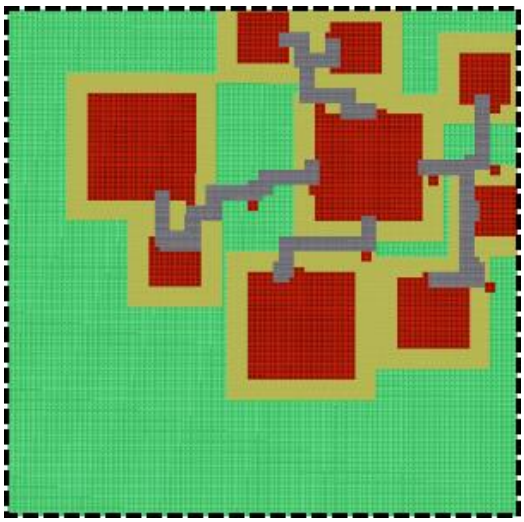

Figura 38 – Representación del mapa, donde se observan las parcelas (Rojo) y los caminos que las unen (Gris).

4.4 Sistema de guardado

Para el almacenamiento y lectura de información se utilizan archivos *JSON*, puesto que su estructura como diccionario se adapta perfectamente a las necesidades del proyecto.

4.4.1 Opciones

Se tienen dos archivos con configuraciones que contienen variables importantes, de esta forma se pueden modificar las opciones de un asentamiento sin tener que volver a compilar el código, únicamente cambiando estos archivos.

El primero, *PreprocessingOptions.json* contiene la información para la parte de preprocesado.

PENALTY_THRESHOLD: Desviación media máxima que puede tener un edificio para que interese guardar la parcela.

MIN_BLOQUES_REQ: Número mínimo de bloques que debe tener una parcela para no ser eliminada durante el preprocesado.

MIN_WIDTH_PLOT_REQ: Ancho mínimo que debe tener una parcela para no ser eliminada durante el preprocesado.

MIN_LENGTH_PLOT_REQ: Alto mínimo que debe tener una parcela para no ser eliminada durante el preprocesado.

BLACK_LIST: Lista con la ID de los bloques que se eliminan durante el preprocesado.

WHITE_LIST: Lista con la ID de los bloques que se ignoran durante el preprocesado.

SettlementOptions.json contiene las variables para el módulo de la creación de asentamientos.

NUM_PLOTS: Número máximo de edificios que puede haber en un asentamiento.

PADDING: Número de bloques de separación entre edificios.

CLUSTER_BASE_DIST: Distancia base entre las agrupaciones de caminos.

CLUSTER_DIST_MULTIPLIER: Factor de crecimiento de la distancia máxima para agrupar sets en cada iteración.

CLUSTER_BASE_MAX_SETS: Número de elementos máximo que tiene un set en la primera iteración.

CLUSTER_SETS_MULTIPLIER: Factor de crecimiento del número de elementos máximos de los sets en base a "CLUSTER_BASE_MAX_SETS".

PATH_DIAGONAL: Booleano que define si pueden construir caminos en diagonal.

RAMP_PENALTY: Penalización por desnivel cuando se ejecuta el algoritmo A Estrella.

DRAW_MAP: Booleano que determina si se va a pintar el mapa de depuración.

LAMP_PAD: Distancia mínima entre las farolas.

DEFAULT_TREES: Archivos con los árboles que se construyen por defecto en un terreno sin bioma.

TREES_MAX_NUM: El número máximo de árboles que se pueden construir en el asentamiento.

4.4.2 Guardado de edificios

Los edificios individuales se guardan en un archivos *.JSON* con su mismo nombre. Para crear un edificio es necesaria una con una matriz tridimensional de IDs de cada bloque. Con ayuda de un comando propio, se definen las dimensiones donde se iniciará la creación de la estructura en Minecraft, esto crea una "caja" que sirve como límites para saber dónde se puede construir la estructura. A continuación, se colocan todos los bloques que definen a un edificio, y una vez construido, se ejecuta el comando */save* que guarda la matriz de IDs y sus dimensiones, así como la posición donde está colocada la puerta. Si no se define ninguna posición para la puerta esta se coloca en el centro de la pared que está mirando al norte. Toda esta información se escribe en un archivo *JSON* con el nombre del edificio.

Buildings.json es un archivo que contiene una lista con la información de cada uno de los edificios. Guarda los parámetros relevantes que se necesitan conocer en la parte de preprocesado, y una lista con el nombre de los archivos *prefabs* que pertenecen a ese tipo de edificio. Por ejemplo, un edificio de tipo casa tiene guardadas los archivos con las distintas variaciones que se pueden construir.

4.5 Comandos de Minecraft

En *Minecraft* es posible abrir una consola y utilizar comandos que permiten modificar el estado de la partida y del servidor de distintas formas. En *Forge*, estos comandos actúan como el punto de entrada para la ejecución del código desarrollado. Se ha necesitado de la creación de distintos comandos para implementar funcionalidades que facilitan el desarrollo de la herramienta. Todos los comandos siguen una estructura sintáctica como la siguiente:

"/<nombreDelComando> <parámetro 1> ... <parámetro n>"

A la hora de definir un nuevo comando se debe crear una nueva clase que implementa la interfaz de los comandos. Se puede determinar el número de parámetros que recibe, así como la información que aporta en cada paso de la ejecución del comando. Finalmente, para poder utilizarlo hay que registrarlo en los eventos del servidor en una clase que se encarga de estos llamada *CommandHandler*.

4.5.1 Construir el asentamiento

El comando */BuildSettlement* ejecuta el código correspondiente para generar el asentamiento, se le pasan los parámetros de tamaño del cubo que limita la zona de construcción. Este comando es exigido por la *GDMC* con el fin de unificar la entrada de todos los participantes.

4.5.2 Guardar un edificio

/SaveBuilding es el comando para crear y guardar los edificios. Primero se pasa como parámetros las dimensiones límite para construir. Esto genera una caja que en el mundo se representa creando bloques de oro en los vértices del cubo para que sea fácilmente reconocible la zona donde se va a guardar. Hay que ejecutar el parámetro *"save"* con el nombre del edificio para que todos los bloques que se hayan creado dentro de la caja se guarden en el archivo de texto.
Tiene una funcionalidad añadida extra que permite definir la posición exacta en coordenadas locales al edificio donde estará colocada la puerta.

4.5.3 Construir un edificio

/Building es el comando para construir un edificio de forma individual. Se ha utilizado para la depuración y comprobación de los edificios. Como parámetros recibe el nombre del edificio que se quiere construir y su dirección en puntos cardinales.

4.5.4 Editar un edificio

El comando */EditBuilding* ayuda a la edición de un edificio que ha sido previamente creado, así no es necesario volver a crear de cero ni modificar a mano el archivo de texto que contiene su información. Primero se escribe el nombre del elemento que se va a editar y se coloca una copia física en el mapa. Es posible modificar los bloques dentro de sus límites y una vez terminado se ejecuta el comando con el

parámetro para guardar. Esto vuelve a guardar en el archivo toda la información de la matriz sobrescribiendo la anterior.
También se ha añadido una funcionalidad adicional para arreglar los edificios que están mal orientados ejecutando el parámetro *"fixDir"* y la dirección a la que están actualmente mirando. Lo que hace este comando es girar el edificio para que mire al norte teniendo en cuenta su dirección anterior.

5. Preprocesado del terreno

Antes de generar el asentamiento es necesario adaptar el terreno eliminando de él las imperfecciones más notorias como pueden ser pequeños huecos, vegetación y demás (Figura 39), de esta forma las siguientes fases de generación del asentamiento cuentan con más terreno útil para establecer sus elementos al obtener una superficie similar a la que se muestra en la Figura 40.

Con este objetivo se ha creado un algoritmo que recorre la matriz de bloques por capas y mediante el uso de conjuntos disjuntos realiza distintos tratamientos a los bloques.

Para ello se utiliza una clase basada en conjuntos disjuntos que se encarga de agrupar los bloques iguales adyacentes y calcular su tamaño (que además es el tamaño de las parcelas que se forman), el X mínimo y máximo del conjunto y el Z mínimo y máximo. Gracias a esto se obtiene el ancho y largo de cada conjunto formado, siendo esta información muy útil a la hora de eliminar parcelas con formas poco convenientes de cara a la construcción del asentamiento.

A la hora de realizar el preprocesado, en primer lugar, se realiza un recorrido de la matriz de bloques que se encarga de eliminar todos los bloques cuyas IDs aparecen en una *BLACK LIST*, el objetivo es borrar plantas y demás bloques que pueden dificultar la construcción del asentamiento. Esta *BLACK LIST* es configurable con el fin de probar el impacto que causa en el asentamiento el hecho de borrar distintos tipos de bloques.

Tras esto, se recorre de nuevo la matriz de bloques con el propósito de eliminar pequeñas masas de agua que puedan perjudicar al futuro asentamiento, para lograrlo se obtienen los conjuntos de bloques de agua adyacentes y se convierten en tierra los que no alcancen un tamaño mínimo.

A continuación, se recorre una vez más la matriz de bloques con el fin de obtener los conjuntos de bloques de aire adyacentes, es decir, encuentra parcelas de aire y si estas no alcanzan un tamaño mínimo se sustituyen estos bloques de aire por bloques con una ID calculada siguiendo el siguiente procedimiento: Por cada bloque que se debe sustituir se realiza un recorrido de sus 4 bloques adyacentes, si no son aire en meten en una lista y se selecciona uno aleatorio entre ellos, esta será la nueva ID del bloque de aire. Si todos los bloques adyacentes son aire se le asignará al bloque la ID del bloque que se encuentre una casilla por debajo.

Una vez se han rellenado los huecos de aire se realiza un nuevo recorrido de la matriz que se encargará de obtener los conjuntos de bloques distintos de aire, tras esto se convertirán los bloques de parcelas que no alcancen un tamaño mínimo en bloques de aire, de esta forma se eliminan pequeños fragmentos del terreno que pueden dificultar la colocación de los distintos elementos del asentamiento. Para evitar la eliminación de ciertos tipos de bloques, se utiliza una *WHITE LIST* de IDs configurable, toda ID que aparezca en ella no se tendrá en cuenta a la hora de realizar el suavizado del terreno, por lo que no se realizará ningún tratamiento sobre estos bloques.

Tras numerosas pruebas se detectó que pueden aparecer parcelas de bloques que aun cumpliendo el mínimo número de bloques no son válidas ya que por su forma causan molestias a la hora de establecer el asentamiento.
Para solucionar esto durante el último recorrido de matriz también se eliminan las parcelas de bloques distintos de aire que no alcancen un ancho o largo mínimo, aunque si tengan el mínimo de bloques requerido. (Por ejemplo, parcelas con la forma 1x30.)

Durante el desarrollo del algoritmo se planteó realizar un tratamiento sobre los bloques de lava con el fin de eliminar únicamente los lagos de lava que podrían destruir el asentamiento, pero conservar por ejemplo las cascadas de lava del terreno ya que aportan mucho valor al mismo. Esta idea se desechó debido al comportamiento de los bloques de lava en Minecraft.

Figura 39 – Terreno antes del preprocesado.

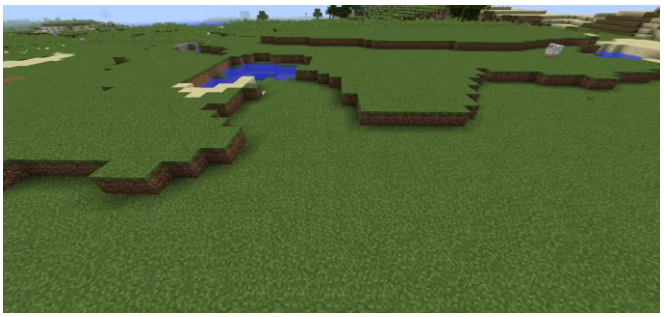

Figura 40 - Terreno tras aplicar el preprocesado.

6. Planificación del asentamiento

Una vez preparado el terreno, se planea la distribución de los distintos elementos. Los diferentes parámetros que intervienen en esta etapa de la construcción del asentamiento son editables desde su respectivo fichero, permitiendo así una forma flexible y cómoda de ajustar los distintos algoritmos.

A grandes rasgos, esta etapa se compone de dos algoritmos principales:

- El escaneado de parcelas, encargado de puntuar las características topográficas (Desnivel, altitud media, etc.) de cada parcela posible.
- La planificación de parcelas, que, utilizando la información obtenida por el algoritmo anterior, selecciona las parcelas optimas, atendiendo a ciertos criterios (distancia al centro, proximidad a otros edificios, altitud y desnivel medios).

6.1 Escaneado de Parcelas

Primer modelo (*Sliding Window* y Cola de Prioridad):

En un primer concepto, se define un tamaño de parcela máximo y mínimo.
A partir de ahí, para cada tamaño de parcela que hay entre los dos valores se escanea la matriz de alturas para cada bloque.

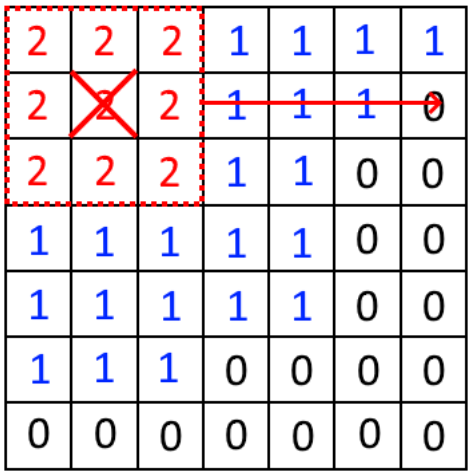

Figura 41 - Visualización del desplazamiento de una "ventana" de 3x3

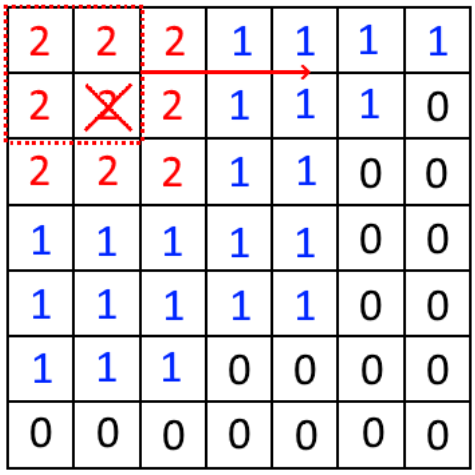

Figura 42 – Visualización del desplazamiento de una "ventana" de 2x2.

Se escoge un "pivote" (La X en las imágenes de la Figura 41 y la Figura 42) y se desplaza el cuadrado del tamaño correspondiente.

Para cada desplazamiento se compara el desnivel de las casillas de la parcela respecto a la altura del pivote, y se calcula la "penalización de pendiente" (Figura 43 y Figura 44) sumando todos estos desniveles.

En la Figura 43 la parcela tendrá una penalización de 0, mientras que en la parcela de la Figura 44 la penalización sería de -3.

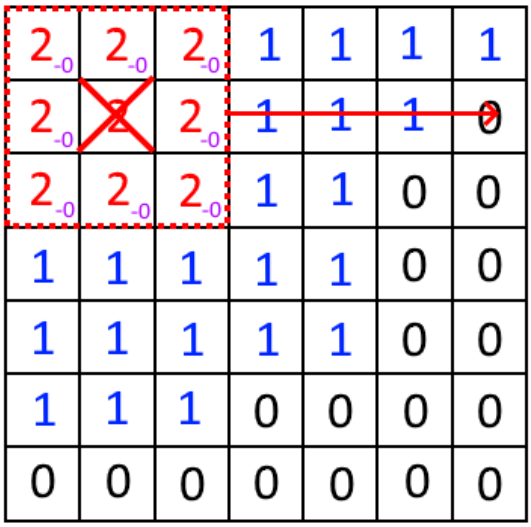

Figura 43 – Simulación del algoritmo en el paso 1. Las penalizaciones individuales representadas en magenta.

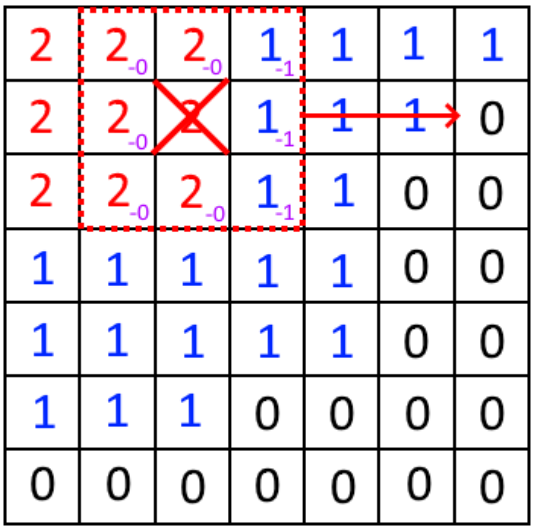

Figura 44 - Simulación del algoritmo en el paso 2. Las penalizaciones individuales representadas en magenta.

Finalmente, cada parcela generada se guarda en una cola de prioridad en función de esta penalización y del tamaño de la parcela, de tal modo que entre dos parcelas con 0 de penalización se priorice la que más grande sea.

Segundo modelo:

En base al anterior algoritmo, se desarrolló otra versión similar. La idea es la siguiente: en vez de guardar todas las combinaciones de tamaño de parcelas en un rango y guardar eso en una cola de prioridad, escanear solo los tamaños que tendrían los edificios y almacenar esta información en una lista para cada edificio:

Esto también cambió la forma de almacenar los edificios, requiriendo que para cada tipo de edificio ahora se almacenará también su ancho y largo, resultando en el siguiente modelo de guardado.

```
“name”:”House”,
“prefabs”:[
	{
		“file”:”house.json”,
		“width”:5,
		“lenght”:5,
		“padding”:1
	},
	{
		“file”:”house1.json”,
		“width”:5,
		“lenght”:5,
		“padding”:1
	},
]
```

De este modo se guarda la información obtenida en el modelo anterior (*Flat Penalty*), para parcelas de 5 x 5 y de 10 x 10.
Además, a la hora de guardar la parcela también se añade como información adicional el archivo al que pertenece la parcela. Así más adelante la parcela sabrá que archivo tiene que usar para construirse.

Una vez acabado el recorrido obtendremos tantas listas como tipos de edificios (*House, Market, Church…*), que tienen la información de las distintas parcelas (*house1, house2, etc).*

En base a los recorridos anteriores, se ha cambiado la estructura del siguiente modo:

1. En el primer recorrido se apunta la altura de todos los bloques, para luego hallar la altura media de la parcela.

2. Después de esto se realiza otro recorrido donde para cada bloque se calcula la desviación de la altura respecto a la altura media. Durante este recorrido también apuntamos cual es la desviación máxima hacia arriba y hacia abajo. En estos casos el agua se considera una penalización de -50 de altura.

3. Cuando ya se ha completado este recorrido se calcula también cuál es la desviación media, y si entra dentro del umbral definido para cada edificio, se guarda la parcela.

Optimización

Con los algoritmos explicados anteriormente se observó que el grueso del tiempo de generación se encontraba en el escaneo, llegando a tardar 8 minutos solo en el escaneo de la iglesia, y alcanzando un total de 16 minutos para escanear todos los edificios. Siento el tiempo límite de la competición 10 minutos, se realizaron los siguientes cambios:

- **Escaneo por tipo:** Se observo que era común que dos o más edificios de un mismo tipo tuvieran el mismo tamaño, pero dada la naturaleza del algoritmo, se escanean por separado.
  Para evitar esto se ha modificado el algoritmo para que, en vez de escanear cada edificio existente, se escaneen solo los distintos tipos de edificio.

Este cambio repercute en dos aspectos del desarrollo:

- Donde antes se almacenaba la información relativa al tamaño del *prefab* en el propio contenedor, ahora se almacena una vez por tipo, como se puede observar en la siguiente tabla:

“name”:”House”, “prefabs”:[{ “file”:”house.json”, “width”:5, “lenght”:5, “padding”:1 }, { “file”:”house1.json”, “width”:5, “lenght”:5, “padding”:1 },] *Antiguo sistema de guardado. Se puede observar que cada prefab contiene la información propia a este.*	“name”:”SmallHouse”, “width”:5, “lenght”:5, “prefabs”:[{ “file”:”house.json”, }, { “file”:”house1.json”, },] *Nuevo sistema de guardado, donde las informaciones propias al prefab se comparten entre las distintas variaciones*

- Esto limita que todos los *prefabs* creados para un tipo de edificio se vean limitados por el tamaño de este. Por ejemplo, todas las casas pequeñas se tratan como una parcela de 10x10.

Para comprobar la eficacia de estos cambios se ha ejecutado el algoritmo dos veces, una antes y otra después de la optimización, obteniendo los siguientes resultados:

Parte [PlotScanning_Church] [0,8419] Parte [PlotScanning_SmallHouse] [0,8850] Parte [PlotScanning_LargeHouse] [0,9134] Parte [PlotScanning_Market] [0,9220] Tiempos de ejecución antes de optimizar	Parte [PlotScanning_Church] [0,6735] Parte [PlotScanning_SmallHouse] [0,6755] Parte [PlotScanning_LargeHouse] [0,6861] Parte [PlotScanning_Market] [0,6945] Tiempos de ejecución tras la optimización. Se puede observar que los edificios con un solo prefab (Como el mercado) no varían su tiempo de ejecución (en ambos casos se ejecuta en 0,0084 sec)

- **Factor de desplazamiento:** Debido a la complejidad cuadrática del algoritmo de escaneo, edificios de mayor tamaño repercuten en gran medida en el tiempo de procesado. Como se comenta anteriormente, el procesado de la Iglesia (39x39) en un mapa de 256x256 suele tardar alrededor de 8 minutos.

  Para reducir el tiempo de las parcelas más grandes se ha diseñado un sistema de detalle, que permite configurar desde el propio documento el nivel de detalle que tendrá el escaneo de ese tipo de edificio.

  Como se explicaba anteriormente, el algoritmo "sliding window" procesa el área del edificio concreto, deslizando bloque a bloque esa área para cubrir todo el mapa.

  Como en edificios cuya área es muy extensa, la diferencia generada de moverse una posición es muy pequeña, se ha creado un "factor de desplazamiento" para configurar cuantas posiciones se desplaza esta área.

  Para comprobar la efectividad de esta mejora, se ha ejecutado el algoritmo dos veces, la primera sin alterar los factores de desplazamiento, y la segunda, aumentando este factor a 3 bloques. Esto genera los siguientes tiempos de procesado:

Parte [TerrainFlattening] [0,5035] Parte [PlotScanning_Church] [8,34624] Parte [PlotScanning_SmallHouse] [8,59707] Parte [PlotScanning_LargeHouse] [12,25026] Parte [PlotScanning_Market] [14,28676] Parte [PlotScanning_Blacksmith] [15,40383] Parte [PlotScanning_Farm] [16,52259] Tiempos de procesado utilizando un factor de desplazamiento de x1 en la iglesia	Parte [TerrainFlattening] [0,5308] Parte [PlotScanning_Church] [1,1181] Parte [PlotScanning_SmallHouse] [1,25764] Parte [PlotScanning_LargeHouse] [4,44129] Parte [PlotScanning_Market] [6,43425] Parte [PlotScanning_Blacksmith] [7,52827] Parte [PlotScanning_Farm] [9,1856] Tiempos de procesado utilizando un factor de desplazamiento de x3 en la iglesia

  Como se observa en la tabla, esta mejora reduce la cantidad de tiempo empleado por las parcelas más grandes.

Combinando ambas técnicas se logra reducir notablemente el tiempo total de ejecución, como se puede apreciar en la siguiente tabla:

```
Parte [ TerrainFlattening ] [ 0,6503 ]
Parte [ PlotScanning_Church ] [ 0,6735 ]
Parte [ PlotScanning_SmallHouse ] [ 0,6755 ]
Parte [ PlotScanning_LargeHouse ] [ 0,6861 ]
Parte [ PlotScanning_Market ] [ 0,6945 ]
Parte [ PlotScanning_Tavern ] [ 0,7024 ]
Parte [ PlotScanning_Blacksmith ] [ 0,7063 ]
Parte [ PlotScanning_Farm ] [ 0,7113 ]
Parte [ PlotScanning_VeggieFarm ] [ 0,7136 ]
Parte [ PlotGraph ] [ 0,8235 ]
Parte [ Clustering ] [ 0,8235 ]
Parte [ Carreteras ] [ 0,9978 ]
Parte [ Direccionamiento ] [ 0,9978 ]
Parte [ Construccion ] [ 1.4287 ]
Parte [ Lamparas ] [ 1.4533 ]
Parte [ Mapa ] [ 1.4634 ]
Ejecución finalizada [ 1.4634 ]
```

6.2 Planificación de Parcelas

Una vez analizado el terreno, es necesario elegir cuales de las posibles parcelas generan un resultado óptimo.

Este algoritmo va fuertemente unido al *Escaneado de las parcelas*, es por ello por lo que cuando se cambió radicalmente el modelo del anterior, se consideró crear un nuevo modelo de este.

Primer modelo:

Para escoger la disposición del asentamiento se usan tres elementos: La cola de prioridad creada anteriormente, un número de parcelas máximo (importado de *options.json)* y una matriz de booleanos para marcar el terreno ocupado. Esta matriz se inicializa con todos los valores a *false* y cada vez que se coloca una parcela se marca ese terreno a *true*, indicando que ha sido ocupado.
En esta primera versión del algoritmo se extrae de la cola de prioridad el primer elemento y se ve si se puede colocar, (es decir, ningún bloque de su área está ocupado por otra parcela) si es posible se coloca y se refleja en la matriz, finalmente se aumenta el número de parcelas situadas.
Se repite este procedimiento hasta que se haya colocado el número máximo de parcelas o ya no se puedan colocar más.

Segundo modelo:

Lo primero que se hizo fue mover este algoritmo al módulo de *Planificación del Asentamiento*, por un lado, para diferenciar bien el escaneado del terreno, que es independiente al algoritmo de construcción.

En este modelo se opera sobre las listas de parcelas obtenidas por el segundo modelo del *Escaneado de Parcelas*.

Para la elección de los distintos elementos se define una función comparadora con el fin de ordenar las parcelas según su valor.
El valor de las parcelas consiste en la suma de distintas variables que afectan a la parcela. Estas variables son:

- **Altura:** Usado para comparar la altitud media de cada parcela.

- **Desviación Media:** Campo que surge en el escaneo, y es usado para priorizar que se escojan parcelas con menor desnivel.

- **Distancia a los focos:** Se utiliza para simular la riqueza de los habitantes del asentamiento. Este atributo es muy útil para estimular el realismo del asentamiento.

Por otro lado, ahora se colocan los edificios en distintas fases. Primero se eligen los focos del asentamiento, *(mercado e iglesia)* y luego se colocan las casas y otros edificios. Esto permite que los edificios más importantes escojan sus mejores emplazamientos mientras que otros edificios menos importantes se seleccionen más tarde, donde la diferencia entre la localización de la parcela afecta menos a la disposición general del asentamiento.

7. Edificación del asentamiento

Una vez se tiene la estructura que define la posición donde se colocan los elementos del asentamiento, estos se representan colocando bloques con la información de sus propiedades en la matriz tridimensional que compone el mundo de *Minecraft*.

7.1 Construcción de edificios

Una vez se han establecido las localizaciones de las parcelas del asentamiento se comienza la edificación de los distintos elementos que se colocarán en ellas.

Cada objeto *plot* tiene una variable con el nombre del archivo que guarda la información del edificio que se va a construir. También contiene un cubo que define el tamaño de la parcela donde se coloca el edificio. Cuando se ejecuta el método para construir el edificio primero se lee la información del JSON correspondiente, extrae la matriz tridimensional y sus dimensiones. Conociendo el tamaño de la parcela y del edificio se ajusta la posición donde se va a colocar el primer bloque para que el edificio se construya siempre en el centro de las parcelas. Por último, se recorre toda la matriz elemento a elemento de abajo hacia arriba y crea el bloque según la ID recibida.

Las clases *plot* tienen un campo que define su orientación según los 4 puntos cardinales, es posible rotar la matriz de un edificio para que al construirse esté orientada en cualquiera de las 4 direcciones. Para ello se aplican a los elementos de la matriz, además de modificar su posición también se tiene que cambiar otra información relevante como la propia orientación individual de cada cubo.

Los bloques de Minecraft no solo tienen la información de su identificador, además poseen propiedades que enriquecen la variedad y posibilidades. Las propiedades que puede tener un bloque son su material, variante, dirección, color, estado de activación, estado de apertura, contenido, forma, eje de rotación, si es medio bloque a qué mitad pertenece, la posición de las bisagras, etc.

Para poder utilizar estas propiedades se hizo que la matriz dejara de ser únicamente de enteros para convertirse en una matriz donde cada elemento contiene la información completa de cada bloque, el entero con su ID y sus múltiples propiedades. Esto nos permite orientar y crear bloques con propiedades complejas. De esta forma los edificios resultantes son mucho más completos e interesantes que los que se tenía en un principio.

7.2 Instanciar entidades

En Minecraft es posible instanciar entidades con comportamiento propio, como aldeanos o varios tipos de animales distintos, esto permite otorgarle "vida" a los asentamientos.
Las entidades comparten el sistema de identificación mediante IDs al igual que los bloques de la Figura 45, para colocarlos la posición se puede definir con valores decimales a diferencia de los bloques cuya posición viene dada por enteros que se corresponden con su posición en la matriz tridimensional que define el mundo.
Una propiedad interesante que poseen es la capacidad de añadirles ítems. La posibilidad combinatoria de entidades e ítems es muy elevada, pero para el proyecto la mayoría son poco relevantes, además muchos de los bloques que nos proporciona *Minecraft* no tienen ninguna utilidad en nuestros asentamientos por lo que definimos una pequeña lista con bloques que no se van a usar y su correspondencia con una entidad.
De esta forma cuando lee uno de estos bloques reservados para entidades en lugar de crear el bloque instancia una entidad con parámetros e ítems preestablecidos. Así es posible crear entidades en coordenadas concretas y a su vez tener esta información en la misma matriz que los bloques.

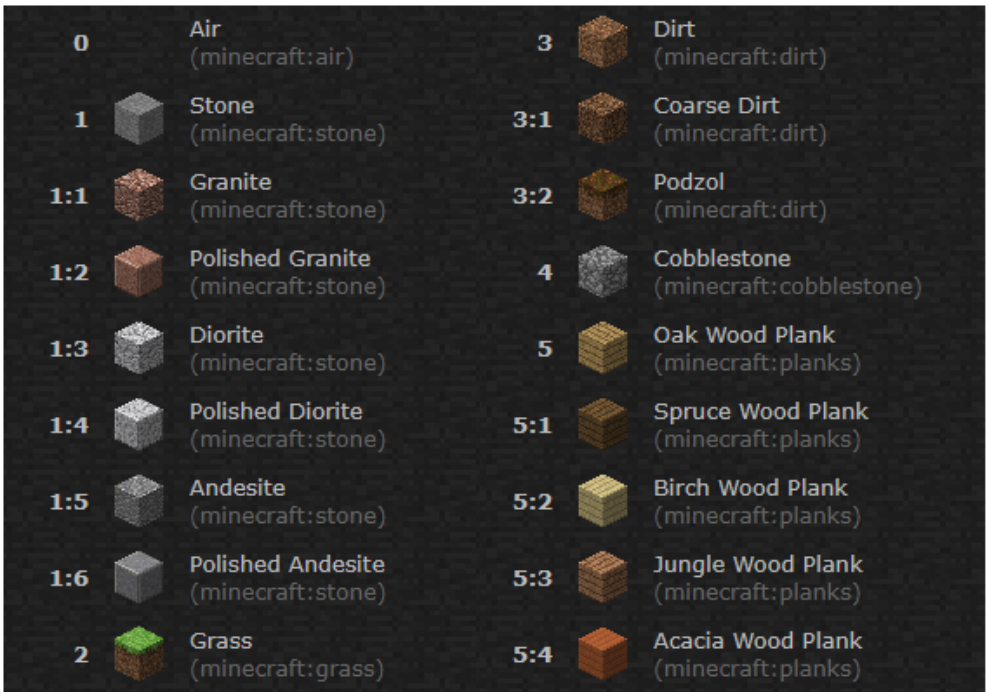

Figura 45 - Ejemplo de la correspondencia entre las IDs y los distintos tipos de bloques.

(https://minecraft-ids.grahamedgecombe.com/)

7.3 Generación de caminos

Para lograr un asentamiento realista es necesario establecer vías de comunicación entre los distintos elementos de este, para ello una vez colocadas las parcelas se crearán caminos que las conecten entre ellas y a los puntos más relevantes (mercado, iglesia, etc.).

7.3.1 Agrupamiento por clústeres

Las parcelas del asentamiento se agrupan con el fin de planificar por donde pasan los distintos caminos y que elementos unen, de esta forma se obtienen caminos más coherentes y útiles ya que unirán varias parcelas.

Para ello se organizan las parcelas en sets cada vez más pequeños como se puede apreciar en el esquema de la Figura 46.
En primer lugar, se visita cada parcela individual y se buscan las parcelas cercanas a ella (que se encuentren a una distancia X definida en las opciones), estas parcelas formarán parte de un mismo set. Los sets tienen un tamaño Y máximo, es decir, cuando el set esté lleno ninguna parcela podrá pertenecer a él, aunque se encuentre a una distancia adecuada.
Cuando un set ha terminado de unir a sus vecinos, se calcula cual es el punto medio entre todos los elementos del set, esta información es necesaria para calcular la distancia hasta el set y para el algoritmo que construye el camino.

Tras esto se itera continuamente sobre los sets que se han construido hasta que el set principal (el mercado) llega al máximo número de sets permitidos, este máximo se define en las opciones.

En cada iteración se aumenta la distancia máxima en la que se buscar parcelas y se reduce el tamaño máximo de los sets (estos factores se definen en las opciones) hasta un mínimo de 2.

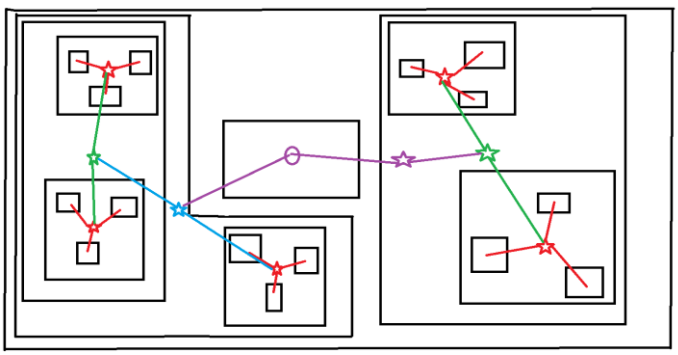

Figura 46 – Esquema de la formación de clústeres. Cada color representa un nivel de agrupamiento.

7.3.2 Construcción del camino

Una vez se han creado todos los clústeres se procede a generar el camino, para ello se ha utilizado un algoritmo basado en A* sobre el mapa del terreno. Para evitar generar caminos con una pendiente muy pronunciada, se establece una fórmula de penalización de desnivel. El factor con el que escala este desnivel viene dado desde el archivo de configuración, permitiendo ajustar el valor durante las pruebas para hallar el valor óptimo.

En un primer momento se aplicaba el algoritmo A* de forma recursiva, uniendo los elementos más pequeños con el centro del grupo superior en la jerarquía.

Mediante este método se obtenían resultados muy caóticos, donde varios caminos se superponen como se puede ver en la Figura 47.

Figura 47 - Generación caótica de caminos. Se puede observar como varios caminos se cruzan entresi, generando un entramado complicado donde no se distinguen las distintas calles.

Para solucionar esto se decidió primero construir los caminos de los niveles más altos, y luego unir los de las ramas inferiores. Además, se aumentó la funcionalidad del algoritmo A* para que cuando encontrara una posición que tuviera la información "Camino" diera por concluida la búsqueda.

Gracias a esto el resultado evolucionó a una forma más ramificada, donde los caminos más pequeños convergen en los más grandes, dando un aspecto más ordenado como se puede ver en la Figura 48 y Figura 49.

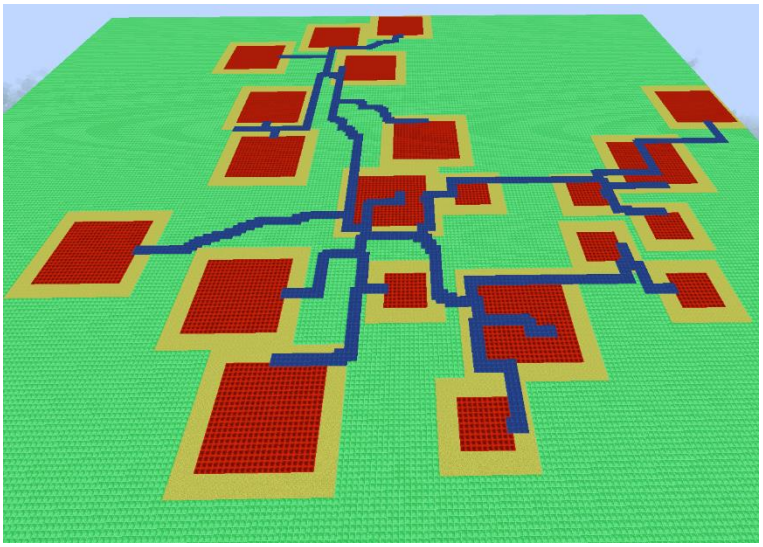

Figura 48 - Disposición final de los asentamientos, los caminos están representados con el color azul, los edificios con rojo, y la separación entre ellos con amarillo.

Figura 49 – Asentamiento correspondiente al mapa de la Fig. 51.

Con el fin de dotar al asentamiento de mayor realismo los caminos que pasan por el agua son sustituidos por puentes, estos cuentan con estructuras que marcan su principio y final y se colocan siguiendo el siguiente criterio:

- Si no hay puente y el camino pasa por un bloque marcado como agua se coloca un principio de puente.

- Si hay un principio de puente y el camino pasa por agua se coloca interior de puente.

- Si hay un principio de puente y el camino no pasa por agua se coloca el fin del puente.

La Figura 50 y Figura 51 muestran ejemplos de los resultados al crear un camino sobre el agua generando puentes.

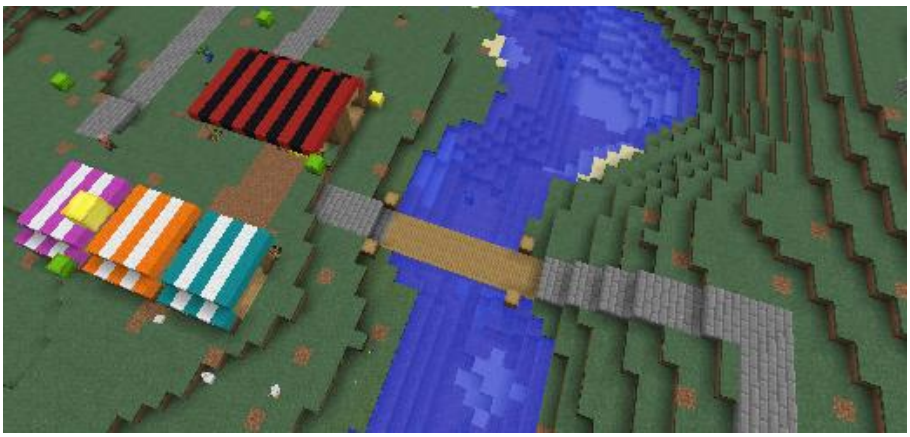

Figura 50 - Puente atravesando un río.

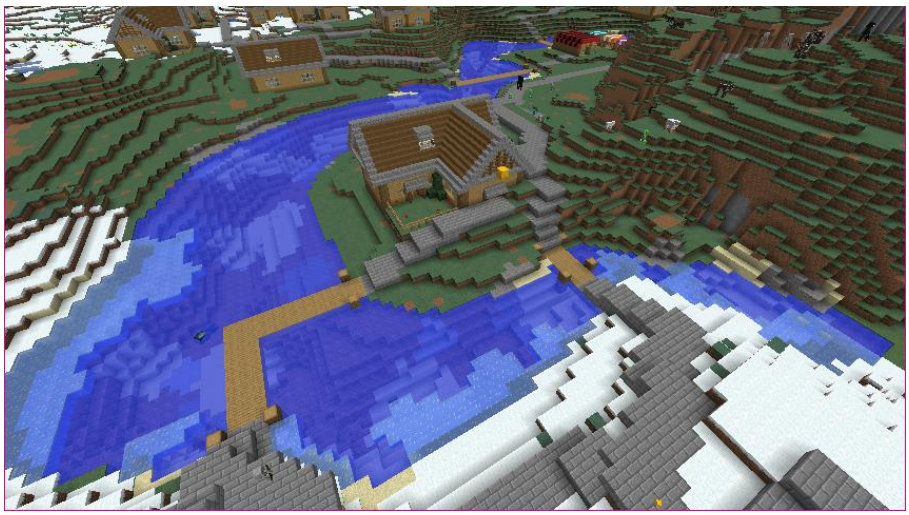

Figura 51 - Dos puentes atravesando un río.

7.3.3 Orientación de los edificios

Una vez elegida la distribución de los caminos es necesario orientar los elementos del asentamiento en función de estos.
En el momento de crear los *prefabs* de los edificios, se unifican las orientaciones para que sus puertas estén mirando siempre al norte, y así poder manejar la rotación de todos de la misma forma.

Para poder definir la orientación de los edificios es necesario saber la posición de su puerta y del punto de conexión con su clúster padre. Se recorren de manera recursiva todos los clústeres, cuando se encuentra uno de tamaño 1 quiere decir que es un edificio, es posible saber cuál es la dirección correcta que se le asignará

al *plot* calculando la diferencia entre la posición del clúster padre y la del propio edificio. De esta forma, si el clúster padre se encuentra a la izquierda, el propio edificio deberá rotar para que al construirse aparezca mirando en esa dirección que tomando como referencia las coordenadas del mundo corresponde con el oeste.

7.4 Colocación de elementos decorativos

Una vez generados los edificios del asentamiento se procede a la colocación de diversos elementos decorativos, estos aportan valor estético y dotan de mayor realismo al poblado.

7.4.1 Algoritmo de selección de emplazamiento para farolas

Las farolas se colocan al lado de los caminos, imitando las ciudades reales donde se sitúan en las zonas más transitadas para abastecerlas de iluminación. Es importante tener en cuenta la proximidad entre ellas para que no se coloquen en un lugar que ya está iluminado porque esto gastaría recursos.

Para facilitar esta configuración se han definido una variable modificable en las opciones del asentamiento que representa la separación mínima que tiene que haber entre dos farolas.

El procedimiento es el siguiente:
Primero se escanea la matriz con la información de la superficie, cuando encuentra un camino se crea un objeto farola adyacente solamente si la distancia a otra farola es inferior a la definida en la variable de separación. De esta forma se consigue iluminar todo el camino de forma uniforme respetando el espacio entre farolas como se puede apreciar en la Figura 52.

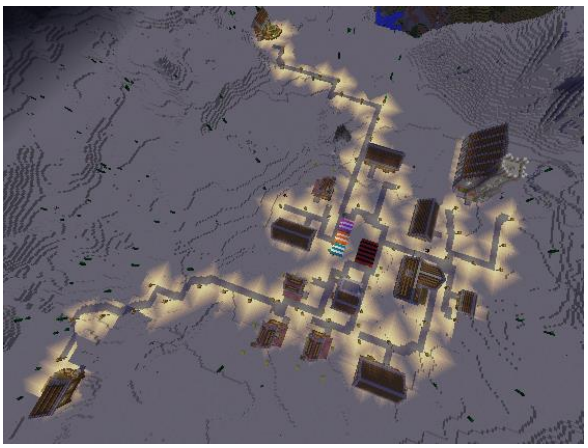

Figura 52 - Farolas repartidas uniformemente junto a los caminos.

7.4.2 Algoritmo de selección de emplazamiento para árboles

Colocar los árboles es el paso final, puesto que son elementos meramente decorativos y colocarlos antes podría entorpecer la construcción de otros elementos más relevantes.

En un principio se pensó colocar los árboles con un algoritmo con la misma funcionalidad que la de las farolas. Se sitúa un árbol en una casilla libre, y el siguiente se coloca en otra casilla libre a una distancia mínima respecto al resto de árboles. Si bien esto funcionaba el resultado eran colocaciones visiblemente artificiales, se barajó aleatorizar la distancia entre los árboles, pero el resultado seguía siendo insatisfactorio.

Finalmente se consideró utilizar un algoritmo mucho menos refinado, pero igual de eficaz y con resultados más naturales. Este consiste en definir un número máximo de árboles que se puedan crear. Se obtiene una posición completamente aleatoria y se intenta construir un árbol, si esa casilla está libre entonces se construye el árbol y se marca la casilla como ocupada. Al definir la altura que va a tener se aplica una pequeña variación aleatoria, así se consiguen distribuciones heterogéneas de alturas que hacen el resultado mucho más orgánico.

Para mejorar la adaptabilidad al terreno se aplica una variación, se obtiene el bioma sobre el que se está construyendo, si el suelo es un bloque normal entonces se escoge aleatoriamente entre un árbol de abeto o uno de roble. Por otra parte, si el bloque donde se va a construir corresponde al bioma de desierto entonces en lugar de construir un árbol normal creará un cactus como se puede apreciar en la Figura 53.

Figura 53 - Árboles sueltos creados en la parte del bioma de bosque, y cactus donde hay suelo desértico.

8. Pruebas

Con el propósito de mejorar la calidad de los asentamientos generados es necesario evaluar los resultados que se obtienen, como no es posible obtener métricas objetivas de ciertas características que se deben tener en cuenta se ha diseñado un cuestionario en el que los entrevistados evalúan ciertas características del asentamiento como su consistencia, coherencia, realismo y estética.

Dicho cuestionario se ha diseñado con el fin de obtener datos de realimentación que ayuden a mejorar los poblados que se generan, además las preguntas se han diseñado teniendo en cuenta el sistema de evaluación que se lleva a cabo en *The GDMC Competition.*

Las encuestas consisten en 3 páginas. Cada una muestra fotos de un asentamiento generado por nuestra herramienta para que los entrevistados lo evalúen según su criterio.

Las preguntas del cuestionario son las siguientes:

- ¿Los edificios del asentamiento se adaptan al bioma en el que se generan?
- ¿El asentamiento es visualmente atractivo?
- ¿Es la estructura del asentamiento consistente? Es decir, ¿Parece que todas las estructuras pertenecen al mismo asentamiento?
- *Valora el número de estructuras diferentes que componen el asentamiento.*
- ¿Consideras realista la disposición de los caminos que unen los elementos del asentamiento?
- ¿Hay alguna característica del asentamiento que lo haga poco realista?
- ¿Añadirías o cambiarías alguna característica del asentamiento?
- A la izquierda se muestra un asentamiento generado por nuestra herramienta y a la derecha uno generado automáticamente en Minecraft. ¿Cuál de los dos prefieres?
- Si es posible, indica por qué prefieres el asentamiento de la pregunta anterior.

Después de presentar la encuesta a un público de 21 sujetos, se han recopilado las respuestas a las distintas preguntas, obteniendo los resultados que se muestran en la Figura 54.

¿Los edificios del asentamiento se adaptan al bioma?	1	2	3	4	5	Total Puntos	Media
Asentamiento 1	0	0	0	8	13	97	**4,619047619**
Asentamiento 2	0	0	2	7	12	94	4,476190476
Asentamiento 3	0	0	1	8	12	95	4,523809524

¿El asentamiento es visualmente atractivo?	1	2	3	4	5	Total Puntos	Media
Asentamiento 1	0	1	1	9	10	91	4,333333333
Asentamiento 2	0	2	0	9	11	95	4,318181818
Asentamiento 3	0	0	0	3	18	102	**4,857142857**

¿Es la estructura del asentamiento consistente?	1	2	3	4	5	Total Puntos	Media
Asentamiento 1	0	1	1	5	14	95	4,523809524
Asentamiento 2	1	0	6	3	11	86	4,095238095
Asentamiento 3	0	0	0	5	16	100	**4,761904762**

Valora el número de estructuras diferentes.	1	2	3	4	5	Total Puntos	Media
Asentamiento 1	0	1	3	8	9	88	4,19047619
Asentamiento 2	0	0	3	6	12	93	4,428571429
Asentamiento 3	0	0	2	6	13	95	**4,523809524**

¿Consideras realista la disposición de los caminos ?	1	2	3	4	5	Total Puntos	Media
Asentamiento 1	0	1	1	9	10	91	4,333333333
Asentamiento 2	0	0	3	5	13	94	**4,476190476**
Asentamiento 3	0	0	3	5	13	94	**4,476190476**

Figura 54 – Puntuación obtenida por cada asentamiento en las preguntas. (De 1 a 5).

Se puede observar, que en general, los asentamientos generados por la herramienta alcanzan los estándares de calidad y estética objetivos, destacando especialmente el Asentamiento 3, que presenta puntuaciones más elevadas frente al resto de asentamientos.
Estos datos son de utilidad a la hora de seleccionar los parámetros a configurar en la generación de futuros asentamientos.

Gracias a los datos reunidos se han podido encontrar características que hacen a los asentamientos generados poco realistas, algunas de estas son:

- Algunos desniveles son extraños, aunque los probadores le restan importancia al parecer asentamientos medievales que no deben tener gran nivel de terraformación.

- Algunas casas están muy alejadas del núcleo del poblado y no tienen demasiados elementos entre ellas.

- Faltan puertos en asentamientos con grandes zonas de agua cercanas.

- Algunos caminos tienen formas algo raras.

- Los colores de la lana del mercado son demasiados vivos comparados con el resto del asentamiento.

Entre las mejoras sugeridas por los probadores se pueden destacar las siguientes:

- Añadir edificios defensivos, murallas y vallas.

- Aumentar la conectividad entre los distintos elementos del mapa y la variedad en los bloques de los caminos.

- Aprovechar más la verticalidad del asentamiento.

- Cambiar los colores del mercado por unos más apagados y detallarlo más.

- Crear nuevos edificios como el puerto, la taberna, etc.

Los encuestados han señalado las siguientes características del asentamiento generado por la herramienta como superiores respecto a los asentamientos generados por Minecraft:

- Poblados más realistas y consistentes, con más complejidad respecto a los de Minecraft.

- Visualmente más bonitos, con mayor variedad de edificios y estructuras con distintos usos como la iglesia, el mercado y la herrería.

- Se adaptan mejor al terreno en el que se construyen por lo que parecen más humanos.

- Asentamientos más naturales, con una distribución de los elementos mejor pensada y no tan sobrecargados como los de Minecraft.
- Más detallados (distintos materiales y colores).
- Se sienten más acogedores.
- Aprovechan mejor la verticalidad del terreno.
- Es más grande e incita a la exploración.
- Simulan la vida de forma más realista con la inclusión de animales, etc.
- Destaca la iluminación nocturna.

Los datos reunidos han permitido ajustar los parámetros que modifican los algoritmos de generación del asentamiento para lograr mejores resultados, además la retroalimentación obtenida es de gran utilidad de cara a mejorar la coherencia y estética del asentamiento, permitiendo diseñar y añadir nuevos elementos en versiones posteriores de la herramienta.

Por otra parte, estos resultados son limitados, ya que no existe un modelo teórico sobre el que evaluar, aun así, aunque no concluyentes, son positivos y aceptables de cara a presentar la herramienta al concurso *"The GDMC Competition".*

9. Discusión

Tras realizar numerosas pruebas utilizando el sistema de generación de asentamientos creado en el proyecto se puede confirmar la hipótesis planteada, es posible generar asentamientos que simulan bases arquitectónicas históricas y que mantienen coherencia entre sus elementos.

Además, los asentamientos que se generan mediante el uso de nuestra herramienta pueden competir en contenido y riqueza visual con los que produce el generador por defecto de *Minecraft*.

9.1 Preprocesado

En este módulo se modifica el terreno generado previamente por Minecraft con el fin de conseguir una superficie óptima, gracias a ello aumenta la cantidad de terreno útil sobre el que se construye el asentamiento a cambio de perder cierta naturalidad en el entorno al eliminar algunas características de este. Esto puede tener impacto de cara a optimizar la adaptabilidad de los asentamientos, aun así, esta pérdida de naturalidad es razonable, ya que si se le diera prioridad se vería reducido el número de elementos del asentamiento y con ello la diversidad de este. Además, la terraformación que se realiza es sutil, asemejándose a la que podría realizarse en asentamientos humanos.

9.2 Técnica de simulación del sistema urbanístico

El sistema urbanístico que se ha seleccionado a la hora de generar los asentamientos es radial, para lograr un resultado similar a este modelo se ha diseñado un sistema de planificación de caminos basado en la agrupación de los distintos elementos del asentamiento.

Como resultado de aplicar esta técnica los asentamientos generados cuentan con una disposición de caminos coherente y que garantiza la conectividad de todos los elementos del mapa sin ramificaciones innecesarias. Sin embargo, la generación de estos caminos puede dar resultados caóticos en mapas muy escarpados, donde se puede perder la transitividad de ciertos segmentos.

Aunque los resultados obtenidos con este modelo son satisfactorios, el centrarse únicamente en él sin tener en cuenta los resultados que podrían producir otros

tipos de modelos si se aplicaran al mismo terreno hace que potencialmente se ignoren soluciones óptimas.
Además, de cara a simular distintas civilizaciones el modelo urbanístico radial que se ha planteado podría no ser el que mejores resultados produzca.

9.3 Límites de la evaluación

Los asentamientos generados deben ser coherentes y visualmente atractivos a ojos humanos, evaluar esto es complicado ya que al tratarse de características subjetivas no existen resultados correctos o incorrectos.

Teniendo esto en cuenta, se ha optado por evaluar las propiedades de los asentamientos generados mediante el análisis de cuestionarios distribuidos a distintos probadores. Ya que no existe una manera objetiva de juzgar las características, en estos formularios realizan las mismas preguntas que se valoran en *The GDMC Competition*, dado que la herramienta se evaluará siguiendo los mismos.

9.4 Elección de plataforma

Se ha escogido *Minecraft* porque permite generar mapas aleatorios coherentes con distintos biomas y superficies, gracias a ello se pueden probar los algoritmos en una gran variedad de situaciones y comprobar su adaptabilidad.

Además, trabajar sobre el mundo es sencillo ya que este se representa como una matriz de cubos, sin embargo, a causa de esto el videojuego presenta una estética vóxel que tiene ciertas limitaciones ya que resta realismo al entorno, aun así, no es una gran desventaja ya que el apartado artístico es consistente.

Otra de las ventajas, es que, al tratarse de un *bestseller*, la comunidad alrededor del juego es muy amplia. Esto facilita la búsqueda de información en lo relacionado a herramientas para desarrollar mods.

9.5 Parametrización de algoritmos

Se ha optado por utilizar el modelo de la herramienta desarrollada ya que es razonable, funcional y los resultados satisfactorios obtenidos justifican el empleo del mismo.

No obstante, debido a la naturaleza del proyecto se pueden valorar distintos enfoques en el desarrollo de los algoritmos, por ejemplo, un modelo basado en técnicas de aprendizaje automático podría dar resultados satisfactorios pero dadas las limitaciones temporales del proyecto no es viable reunir una base de conocimiento fiable.

9.6 Portabilidad y extensibilidad del proyecto

El método utilizado para la creación de construcciones es fácilmente ampliable lo que permite crear nuevos edificios e incluirlos para que aparezcan en el asentamiento, además, generar asentamientos de gran tamaño no tiene gran coste en el tiempo.

Los algoritmos diseñados no trabajan directamente sobre *Minecraft*, sino que procesan matrices tridimensionales y sus valores por lo que sería posible aplicarlos a otros entornos que cumplan estas características, sustituyendo las instrucciones concretas donde se utilizan funciones propias de *Minecraft* que permiten colocar bloques, obtener su ID y crear entidades.

Para extrapolar el proyecto a otro motor cuyos elementos no siguen una organización matricial sería necesario analizar de distinta forma el terreno y adaptar el módulo de construcción para que genere elementos propios de ese entorno.

Además, dada la naturaleza modular del proyecto, permite intercambiar fácilmente los algoritmos usados en cada una de las partes sin alterar el flujo del programa.

Depurar la herramienta es un proceso tedioso puesto que se debe reiniciar la aplicación de *Minecraft* cada vez que se hace una nueva compilación. Para evitar tener que crear un nuevo ejecutable cada vez que se hace un cambio menor se han utilizado ficheros de texto con variables configurables, esto ha agilizado el desarrollo notablemente.

10. Conclusiones y trabajo futuro

A continuación, se expondrán las distintas conclusiones obtenidas a lo largo del desarrollo, así como las mejoras que se pueden realizar sobre el sistema de generación de asentamientos.

10.1 Conclusiones

Durante el proyecto se ha analizado el resultado de generar los asentamientos utilizando el algoritmo en sus distintas etapas, tras realizar numerosas mejoras sobre el algoritmo original se ha conseguido diseñar un algoritmo que genera asentamientos virtuales con resultados satisfactorios ya que es capaz de aprovechar el terreno previamente generado y adaptar sus elementos teniendo en cuenta las características del mapa.

Los asentamientos creados siguen una estética basada en civilizaciones humanas antiguas y la disposición de sus elementos es coherente, considerando los sistemas urbanísticos que se utilizaron como modelo, esto hace que sean funcionales ya que poseen los componentes necesarios para simular las costumbres de una aldea.

El emplazamiento de las casas se basa en el nivel económico de los habitantes, contando el asentamiento con los edificios necesarios para simular sistemas de agricultura, comercio, transporte, religiosos...

En definitiva, el algoritmo de generación de asentamientos cumple las hipótesis y objetivos que se plantearon al principio del proyecto, considerándose apta por la GDMC de cara a participar en su concurso.

10.2 Trabajo futuro

El sistema de generación de asentamientos que se plantea en el proyecto puede ser sometido a mejoras como las planteadas a continuación:

- Desarrollo de un sistema económico complejo que afecte visiblemente a los asentamientos y sus edificios, por ejemplo, mediante la expansión del sistema basado en la distancia de elementos a núcleos urbanos, el desarrollo de *IA* para que los *NPCs* simulen una vida en el asentamiento y puedan influir sobre el mismo, etc.

- Nuevos tipos de asentamientos basados en distintas culturas, implementando las características que las hacen únicas y diferencian sus sistemas urbanos, como el tipo de edificios, los materiales de construcción utilizados o la disposición urbanística, etc.

- Inclusión de edificios con distintas funciones para aumentar la diversidad en el asentamiento, así como añadir variantes para el mismo tipo de construcción. Por ejemplo, se podrían crear edificios específicos para la interacción con áreas acuáticas como muelles, barcos, o zonas de pesca.

- Simulación del paso del tiempo para generar el asentamiento en distintas épocas y, a partir de ello, generar una narrativa compleja que explique y dé sentido a la distribución de los elementos.

11. Conclusions and future work

The conclusions reached during the course of development will be presented below, as well as the improvements that can be made to the system of settlement generation.

11.1 Conclusions

During the project, the result of generating the settlement using the algorithm in its different stages has been analyzed and, after archiving many improvements on the original algorithm, designing an algorithm that creates virtual settlements with satisfactory results due to the ability to leverage the area and adapt its elements being aware of the map characteristics has been accomplished.

The created settlements aim at achieving an aesthetic based on old civilizations and the elements' arrangement is coherent in consideration of the urban systems which were used as a model. This make them functional since they possess every necessary component to simulate the tradition in a village.

The houses position is based on the villagers' economical standards; the settlement also has got the necessary buildings to simulate shops, transport, religious and agriculture systems, etc.

To sum up, the settlement generation algorithm accomplishes the hypothesis and objectives which were suggested at the beginning of the project, being considered competent by the *GDMC* in order to participate in its contest.

11.2 Future work

The system of settlement generation proposed in the project can be subject to improvements such as those outlined below:

- Development of a complex economic system that visibly affects the settlements and their buildings, for example, through the expansion of the system based on the distance of elements to urban centres, the development of AI so that NPCs simulate life in the settlement and can influence it, etc.

- New types of settlements based on different cultures, implementing the characteristics that make them unique and differentiate their urban

systems, such as the type of buildings, the construction materials used or the urban layout, etc.

- Inclusion of buildings with different functions to increase the diversity in the settlement, as well as adding variants for the same type of construction. For example, specific buildings could be created for interaction with water areas such as docks, boats, or fishing places.

- Simulation of the passage of time to generate the settlement in different periods and, from this, generate a complex narrative that explains and gives meaning to the distribution of the elements.

12. Contribución personal

Borja Núñez Béjar.

Investigación previa.
Encargado junto a Miguel Ángel Gremo Alcocer y Gabriel Aramis Sardaneta del Collado del estudio previo sobre generación procedimental en videojuegos y la investigación y prueba de herramientas para la generación procedimental en *Minecraft*.

Investigación sobre modelos urbanísticos para el sistema de colocación y organización del asentamiento y el algoritmo de construcción de caminos.

Prototipado en Python.
Desarrollo del prototipo de la herramienta utilizando filtros para *MCEdit* programados en *python.*

Sistema de preprocesado.
Creación de una clase basada en conjuntos disjuntos que permite localizar agrupaciones de bloques adyacentes del mismo tipo mediante un recorrido de la matriz tridimensional y calcular su tamaño, ancho y largo.

Creación de un algoritmo de "*BLACK LIST*" que permite seleccionar para su eliminación bloques molestos o innecesarios de cara a la generación del asentamiento, por ejemplo, los distintos tipos de vegetación.

Inclusión de un sistema de *"WHITE LIST"* configurable con el fin de evitar la modificación de bloques durante el preprocesado del terreno, esto permite seleccionar que tipo de bloques se mantienen inalterados, dotando de mayor naturalidad al asentamiento al conservarse parte del entorno original.

Algoritmo de sustitución de los bloques de aire y agua de parcelas que no llegan a un tamaño mínimo, transformándolos en bloques similares a los adyacentes con el objetivo de mantener la consistencia del terreno.

Algoritmo de eliminación de bloques distintos al aire y agua que no alcanzan un tamaño mínimo o tienen una forma que no es útil de cara a construir el asentamiento, esto permite eliminar, por ejemplo, agrupaciones de tierra o roca que llegan al tamaño mínimo establecido, pero son muy largas o anchas, haciendo imposible edificar sobre ellas.

Sistema de configuración de preprocesado mediante *JSON* permitiendo cambiar las características de este (tamaño, ancho y alto mínimo requerido de las agrupaciones de bloques, ID de los bloques que forman la *"BLACK LIST"* y la *"WHITE_LIST"*, etc.) sin necesidad de recompilar la herramienta, siendo esto útil a la hora de probar distintas configuraciones al generar los asentamientos ya que permite ahorrar tiempo entre pruebas.

Realización de pruebas del sistema de preprocesado utilizando distintas configuraciones sobre diferentes biomas con el fin de obtener superficies óptimas sobre las que construir el asentamiento en el mayor número de escenarios posibles.

Agrupamiento por clústeres.
Diseño y desarrollo del algoritmo de generación del sistema de clústeres que permite agrupar los elementos del asentamiento por teniendo en cuenta la distancia a la que se encuentran con el fin de formar grupos que simulan barrios y evitar la construcción de caminos innecesarios.

Junto a Miguel Ángel Gremo Alcocer se crearon las estructuras necesarias para manejar los grupos, y, de forma recursiva se recorren para los distintos algoritmos que operan sobre ellos.

Caminos.
Algoritmo para la generación de caminos que conecten todos los elementos del asentamiento de forma coherente, utilizando una versión modificada de A* de manera que, si durante la construcción de un nuevo camino este se encuentra con una casilla de un camino anterior la búsqueda concluye termina la generación del camino. Esto permite buscar el camino óptimo transitable entre cada uno de los puntos que se deben unir del asentamiento.
Se ha desarrollado los métodos necesarios para que este algoritmo actúe sobre el mapa generado.
Además, se establecen parámetros en el archivo de configuración externa que permiten ajustar la capacidad del algoritmo para penalizar los caminos cuya pendiente sea elevada.

Sistema de ubicación de puentes a partir de los caminos generados previamente, con el fin de dotar de mayor realismo al asentamiento, cuando los caminos pasan por agua se generan puentes, estos cuentan con un principio y un final que aporta valor estético.

Depuración.
Depuración y corrección de los errores que surgían durante el desarrollo de los distintos algoritmos.

Pruebas.
Creación de formularios, realización de encuestas y estudio de los datos obtenidos con el objetivo de evaluar que parámetros de la herramienta producen mejores resultados.

Documentación.
Realización de la memoria del proyecto junto a Miguel Ángel Gremo Alcocer y Gabriel Aramis Sardaneta del Collado.

Miguel Angel Gremo Alcocer.

Investigación previa.
Encargado junto a Borja Nuñez Béjar y Gabriel Aramis Sardaneta del Collado del estudio previo sobre generación procedimental en videojuegos y la investigación y prueba de herramientas para la generación procedimental en *Minecraft*, así como la investigación correspondiente a los distintos modelos urbanísticos.

Prototipado en Python.
Creación del proyecto en Python,asi como la estructura principal del proyecto.
Comunicación con *MCEdit* y desarrollo de filtros para colocar las parcelas en *MCEdit*.

Set-up del proyecto en Java.
Búsqueda de toda la información necesaria para crear un proyecto de Eclipse con *Forge* y enlazarlo al *Minecraft*. Desarrollo de scripts que automatizan la compilación y ejecución del proyecto.

Montaje de la estructura inicial en Java.
Creación del esqueleto del proyecto.
Los comandos necesarios para la comunicación con *Minecraf*t, y las herramientas necesarias para depurar y colocar bloques de manera rápida.
Creación de los tres módulos base (*Preprocesado, Settlement y Build*), así como la comunicación necesaria entre ellos.

Escaneado de parcelas.
Desarrollo de los distintos modelos del algoritmo de escaneado, incluyendo el volcado de la información generada a un archivo externo. También los distintos modelos del algoritmo, cuyos cambios supusieron un radical cambio en la estructura creada anteriormente.
Durante el desarrollo de esto también se diseñó la estructura de guardado y cargado de los distintos tipos de edificios junto a Gabriel Aramis Sardaneta del Collado.

Representación del mapa.
Desarrollo del sistema de representación del mapa que busca aunar las distintas matrices de información que se iban generando conforme se necesitaban. Esto ayudo a la rápida integración de nueva información sobre el mapa para que pudiera ser fácilmente usable desde otras partes del proyecto.

Lectura y guardado de información.
Inclusión de librería para utilizar archivos *JSON* al proyecto e implementación de la clase encargada de lectura y guardado de los mismos junto con Gabriel Aramis Sardaneta del Collado.

Carga de información.
Carga mediante *JSON* y deserialización de la información guardada en el archivo temporal resultante del algoritmo de escaneado.

Configuración del segundo modulo.
Carga de la configuración necesaria para el módulo de creación del establecimiento. En él se especifican los parámetros que intervienen en la selección de parcelas, la agrupación por clústeres y la creación del camino.

Elección de parcelas.
Desarrollo del algoritmo encargado de crear el grafo correspondiente al pueblo en base a la información generada en el algoritmo de escaneado.
Primer modelo:
Algoritmo de selección de las parcelas más optimas en función a la cola de prioridad generada en el preprocesado.

Segundo modelo.
Desarrollo de los distintos comparadores utilizados en la selección de cada tipo de edificio.
Desarrollo de las distintas fórmulas necesarias para el cálculo de los diversos parámetros utilizados por los comparadores para establecer las parcelas en un orden óptimo.

Ocupación en el mapa.
Encargado de comprobar la disponibilidad de la parcela objetivo en el mapa actual. Una vez se valida la disponibilidad de la parcela, encargado de actualizar la información correspondiente a la parcela, marcando como ocupado el área ocupada por esta, y realizando los cálculos necesarios para añadir el espaciamiento determinado por el archivo de configuración externo.

Finalmente, la selección de las parcelas específicas para cada tipo en base a la combinación entre la ocupación y los parámetros especificados en la configuración.

Agrupamiento por clústeres.
Diseño y desarrollo del algoritmo de generación del sistema de clústeres que permite agrupar los elementos del asentamiento por teniendo en cuenta la distancia a la que se encuentran con el fin de formar grupos que simulan barrios y evitar la construcción de caminos innecesarios.

Junto a Borja Nuñez Bejar se crearon las estructuras necesarias para manejar los grupos, y, de forma recursiva se recorren para los distintos algoritmos que operan sobre ellos.

Caminos.
Algoritmo para la generación de caminos que conecten todos los elementos del asentamiento de forma coherente, utilizando una versión modificada de A* de manera que, si durante la construcción de un nuevo camino este se encuentra con una casilla de un camino anterior la búsqueda concluye termina la generación del camino. Esto permite buscar el camino óptimo transitable entre cada uno de los puntos que se deben unir del asentamiento.
Se ha desarrollado los métodos necesarios para que este algoritmo actúe sobre el mapa generado.
Además, se establecen parámetros en el archivo de configuración externa que permiten ajustar la capacidad del algoritmo para penalizar los caminos cuya pendiente sea elevada.

Durante las iteraciones siguientes se ha mejorado la capacidad transitable del camino, logrando que aquellas pendientes intransitables sean suavizadas.
También se ha mejorado el algoritmo para mover aquellos centros de los clústeres que estuvieran dentro de otros edificios, lo que rompía la conectividad del asentamiento.

Depuración
Depuración y corrección de los errores que surgían durante el desarrollo de los distintos algoritmos

Optimización
Mejora de los distintos algoritmos para lograr el tiempo objetivo marcado por las pautas de la competición, reduciendo el tiempo de procesado de las partes que más tiempo ocuparan (Escaneado de Parcelas y Planificación de Parcelas).

Creación de los *prefabs* utilizados
Diseño y desarrollo de los diferentes edificios que forman el asentamiento.

Pruebas.
Realización de encuestas y estudio de los datos obtenidos con el objetivo de evaluar que parámetros de la herramienta producen mejores resultados.

Documentación.
Realización de la memoria del proyecto junto a Borja Núñez Béjar y Gabriel Aramis Sardaneta del Collado.

Gabriel Aramis Sardaneta del Collado.

Investigación previa.
Encargado junto a Miguel Ángel Gremo Alcocer y Borja Núñez Béjar del estudio previo sobre generación procedimental en videojuegos y la investigación y prueba de herramientas para la generación procedimental en *Minecraft*.
Investigación sobre modelos urbanísticos para el sistema de colocación y organización del asentamiento y el algoritmo de construcción de caminos.

Sistema para la representación gráfica del asentamiento utilizando Minecraft
Encargado principalmente de desarrollar y corregir los errores de los apartados del módulo de representación gráfica de los asentamientos. Esto incluye:

Investigación de la *API* de *Forge* para interactuar con el mundo de *Minecraft*, e implementación de clases propias que facilitan su utilización en el proyecto, como por ejemplo métodos para crear bloques con propiedades complejas e instanciar entidades.
La construcción de edificios a partir de bloques de *Minecraft* cuya información se encuentra en archivos *JSON*.
La rotación de los *prefabs* aplicando una rotación de la matriz tridimensional en la dirección determinada, así como aplicar una rotación individual a las propiedades de cada bloque.
El sistema de puertas, que define la posición de entrada de cada edificio, esta se corresponde con el punto de conexión de los caminos del asentamiento.
Instanciar entidades en el mundo utilizando bloques especiales que marcan el punto donde va a aparecer.
Equipar unidades con ítems del juego.
Inclusión de nuevos tipos de bloques según se necesitaran en el desarrollo para que la herramienta fuera capaz de reproducirlos correctamente con todas sus propiedades.
Construcción de cimientos para los edificios, si el edificio se coloca sobre un desnivel la capa de bloques del suelo se extiende hacia abajo creando así cimientos.
Construcción de los edificios según una "*máscara*" definida por los bloques del suelo, esto hace que solamente se modifiquen bloques del terreno si se encuentran en el interior del edificio.
Dentro del sistema de construcción de edificios definir bloques especiales que deben construirse después del resto para evitar problemas de colocación, por ejemplo, intentar colocar una antorcha antes que la pared que la sostiene.

Caminos
Depuración y búsqueda de errores en la ejecución del algoritmo.
Hacer que los caminos surjan y se dirijan hacia la posición donde se encuentran las puertas de los edificios.

Orientación de los elementos del asentamiento rotándolos según su posición respecto al camino que parte de su puerta.

Decoración del asentamiento
Desarrollo de los algoritmos para colocar elementos decorativos como último paso en la generación del asentamiento, estos son:
La colocación de farolas a lo largo de los caminos para iluminar uniformemente el asentamiento.
Emplazamiento aleatorio de árboles en los lugares aptos del terreno, y sustitución por cactus en biomas desérticos.
Eliminación de todas las entidades que se encuentran en el área antes de empezar la generación del asentamiento.

Comandos en Minecraft
Creación de los comandos para guardar y editar *prefabs* que modifican y almacenan la información de los edificios en archivos *JSON*, así como un comando para construir un edificio individual.
Estos comandos han sido especialmente útiles para la creación y depuración de construcciones propias utilizando *Minecraft*.

Lectura y guardado de información
Inclusión de librería para utilizar archivos *JSON* al proyecto e implementación de la clase encargada de lectura y guardado de los mismos junto con Miguel Ángel Gremo Alcocer.

Prefabs edificios Minecraft
Creados los *prefabs* de las granjas de animales a partir de la base de casas hechas por Miguel Ángel Gremo Alcocer.

Depuración
Depuración y corrección de los errores que surgían durante el desarrollo de los distintos algoritmos

Pruebas
Creación de formularios, realización de encuestas y estudio de los datos obtenidos con el objetivo de evaluar que parámetros de la herramienta producen mejores resultados.

Documentación
Realización de la memoria del proyecto junto a Borja Núñez Béjar y Miguel Angel Gremo Alcocer.

13. Bibliografía

Adams, T. (2015). Simulation principles from Dwarf Fortress. Game AI Pro, 2, 519-521.

Adams, T., and Adams, Z. Dwarf Fortress.

Aitken, M., Butler, G., Lemmon, D., Saindon, E., Peters, D., & Williams, G. (2004). The Lord of the Rings: the visual effects that brought middle earth to the screen. In ACM SIGGRAPH 2004 Course Notes (pp. 11-es).

Amato, A. (2017). Procedural content generation in the game industry. In Game Dynamics (pp. 15-25). Springer, Cham.

Ames, M. G., & Burrell, J. (2017, February). 'Connected Learning'and the Equity Agenda: A Microsociology of Minecraft Play. In Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (pp. 446-457).

Collins, K. (2009). An introduction to procedural music in video games. Contemporary Music Review, 28(1), 5-15.

Compton, K., & Mateas, M. (2006, June). Procedural Level Design for Platform Games. In AIIDE (pp. 109-111).

Duncan, S. C. (2011). Minecraft, beyond construction and survival. Well Played: a journal on video games, value and meaning, 1(1), 1-22.

Ekaputra, G., Lim, C., & Eng, K. I. (2013). Minecraft: A game as an education and scientific learning tool. ISICO 2013, 2013.

Fernández-Vara, C., & Thomson, A. (2012, May). Procedural generation of narrative puzzles in adventure games: The puzzle-dice system. In Proceedings of the The third workshop on Procedural Content Generation in Games (pp. 1-6).

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

Garda, M. B. (2013). Neo-rogue and the essence of roguelikeness. Homo Ludens, 1(5), 59-72.

Green, M. C., Salge, C., & Togelius, J. (2019, August). Organic building generation in minecraft. In Proceedings of the 14th International Conference on the Foundations of Digital Games (pp. 1-7).

Grey, J., & Bryson, J. J. (2011, April). Procedural quests: A focus for agent interaction in role-playing-games. In Proceedings of the AISB 2011 Symposium: AI & Games (pp. 3-10). University of Bath.

Gupta, A., & Gupta, A. (2015). Minecraft Modding with Forge: A Family-Friendly Guide to Building Fun Mods in Java. " O'Reilly Media, Inc.".

Kelly, G., & McCabe, H. (2007, November). Citygen: An interactive system for procedural city generation. In Fifth International Conference on Game Design and Technology (pp. 8-16).

Kelly, G., & McCabe, H. (2006). A survey of procedural techniques for city generation. The ITB Journal, 7(2), 5.

Kerlow, I. V. (2009). The art of 3D computer animation and effects. John Wiley & Sons.

Khalifa, A., Perez-Liebana, D., Lucas, S. M., & Togelius, J. (2016, July). General video game level generation. In Proceedings of the Genetic and Evolutionary Computation Conference 2016 (pp. 253-259).

MCEdit. https://github.com/mcedit/mcedit

Olsen, J. (2004). Realtime procedural terrain generation.

Parberry, I. (2014). Designer worlds: Procedural generation of infinite terrain from real-world elevation data. Journal of Computer Graphics Techniques, 3(1).

Persson, M., & Bergensten, J. (2011). Minecraft. Stockholm, Sweden: Mojang AB.

Ryan, J. O., Mateas, M., & Wardrip-Fruin, N. (2015, November). Open design challenges for interactive emergent narrative. In International Conference on Interactive Digital Storytelling (pp. 14-26). Springer, Cham.

Ryu, D., & Kanyuk, P. (2007, May). Rivers of rodents: an animation-centric crowds pipeline for Ratatouille. In SIGGRAPH Sketches (p. 65).

Salge, C., Green, M. C., Canaan, R., & Togelius, J. (2018, August). Generative design in minecraft (GDMC) settlement generation competition. In Proceedings of the 13th International Conference on the Foundations of Digital Games (pp. 1-10).

Salge, C., Guckelsberger, C., Green, M. C., Canaan, R., & Togelius, J. (2019). Generative Design in Minecraft: Chronicle Challenge. arXiv preprint arXiv:1905.05888.

Shaker, N., Togelius, J., & Nelson, M. J. (2016). Procedural content generation in games. Switzerland: Springer International Publishing.

Short, D. (2012). Teaching scientific concepts using a virtual world—Minecraft. Teaching Science-the Journal of the Australian Science Teachers Association, 58(3), 55.

Thalmann, D., Hery, C., Lippman, S., Ono, H., Regelous, S., & Sutton, D. (2004). Crowd and group animation. In ACM SIGGRAPH 2004 course notes (pp. 34-es).

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