

Distribución de recursos personalizados sobre superficies en  
*Blender*

Jose Daniel Rave Robayo  
Sergio José Alfonso Rojas  
Daniel Illanes Morillas

---



Facultad de Informática  
Universidad Complutense de Madrid

Trabajo de Fin de Grado en Desarrollo de Videojuegos

Agosto de 2023

Directores y colaboradores:

Prof. Dr. Carlos León Aznar  
Alejandro Villar Rubio



# Custom props distribution over surfaces in *Blender*

Jose Daniel Rave Robayo  
Sergio José Alfonso Rojas  
Daniel Illanes Morillas

---



Computer Science Faculty  
Universidad Complutense de Madrid

BCS in Game Development

August 2023

Directors and collaborators:

Dr. Prof. Carlos León Aznar  
Alejandro Villar Rubio

# Agradecimientos

Gracias a nuestras familias, amigos y seres queridos más cercanos. Sin su ánimo no hubiera sido posible superar los desafíos que se presentaron en el desarrollo de este trabajo.

Sentimos una especial gratitud a todos aquellos que se prestaron voluntarios para realizar las pruebas de usuario, dándonos consejos y sugerencias para mejorar.

Nuestro más sincero agradecimiento a nuestros dos tutores, Carlos León Aznar y Alejandro Villar Rubio. Ambos mostraron un inmenso interés para que este proyecto salga adelante, dándonos consejos y guías, que alguna que otra nos puso los pies en la tierra. Gracias a Alejandro, por revisar constantemente la memoria y enseñarnos un poco el mundo académico.

Y por supuesto, gracias a esa personita que permanecía todas las noches haciendo compañía mientras me quejaba.

# Resumen

La distribución de objetos es parte de las principales tareas cuando se trata de crear escenarios virtuales, ya sea para videojuegos o con propósitos de conceptualización. En la actualidad, existen una gran cantidad de herramientas digitales, como programas de modelado 3D, que proporcionan alguna funcionalidad de distribución. Sin embargo, la mayoría de ellas ofrecen una flexibilidad limitada, restringiendo la personalización por parte del usuario y limitándose a un programa específico, lo que impide su uso en cualquier otro contexto. Este proyecto tiene como objetivo desarrollar una extensión extrapolable que proporcione un sistema de distribución de objetos, permitiendo al usuario definir sus propias reglas y personalizar la distribución. En consecuencia, esta propuesta está fuertemente relacionada con la inteligencia artificial y los algoritmos de búsqueda de caminos, para lo cual el sistema se basa en una biblioteca estándar de algoritmos y estrategias de inteligencia artificial para resolver problemas de búsqueda de caminos con múltiples soluciones. Se proporciona, una implementación en *Python* del sistema como parte de la investigación, la cual fue creada para el programa 3D *Blender*. Además, se realizó una evaluación con varios usuarios y un análisis correspondiente para validar parcialmente su usabilidad y la calidad de sus resultados.

**Palabras clave**— Herramienta 3D, Distribución, Extensión, Creación de escenarios, Inteligencia Artificial, AIMA, Blender, Búsqueda de caminos

# Abstract

Object distribution is part of the main tasks when it comes to creating virtual scenarios, whether they are used for a videogame or conceptualization purposes. These days, there are a great number of digital tools, such as 3D modeling software, that provide some sort of distribution functionality. However, most of them offer limited flexibility, limiting user customization, and exclusive to a specific software, preventing their use in any other context. This project aims to develop an add-on that provides an object distribution system, allowing the user to define their own rules and customize the distribution. In consequence, this proposal is strongly related to artificial intelligence and pathfinding algorithms, for which the system relies on some off-the-shelf library of artificial intelligence algorithms and strategies to solve pathfinding problems with multiple solutions. A *Python* implementation of the system is provided as part of the research, which was created for the 3D software *Blender*, along with evaluation by multiple users, and a corresponding analysis to partially validate its usability and the quality of its results.

**Key words**— 3D Tool, Object Scattering, Add-on, Stage Building, Artificial Intelligence, AIMA, Pathfinding, Blender

# Índice general

<b>Agradecimientos</b>	<b>I</b>
<b>Resumen</b>	<b>II</b>
<b>Abstract</b>	<b>III</b>
<b>Índice</b>	<b>IV</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	2
1.2. Hipótesis de partida . . . . .	3
1.3. Objetivos . . . . .	4
1.4. Metodología . . . . .	5
1.4.1. Tecnologías y herramientas usadas . . . . .	5
1.4.2. Plan de trabajo . . . . .	7
1.5. Estructura del documento . . . . .	7
<b>2. Estado del arte</b>	<b>10</b>
2.1. Generación procedimental . . . . .	10
2.1.1. Números pseudo-aleatorios . . . . .	12
2.1.2. Búsqueda de caminos . . . . .	12
2.2. Tecnologías relacionadas . . . . .	13
2.2.1. Modelos 3D y programas de modelado 3D . . . . .	13
2.2.2. <i>AIMA</i> . Inteligencia Artificial: Un Enfoque Moderno . . . . .	14

2.3. Revisión de programas y extensiones relacionados con la distribución de objetos . . . . .	17
<b>3. Arquitectura del proyecto</b>	<b>20</b>
3.1. Módulos del proyecto . . . . .	21
3.2. Estructura de clases . . . . .	21
3.2.1. Módulo Blender . . . . .	21
3.2.2. Módulo Distribución . . . . .	23
3.3. Flujo de ejecución . . . . .	24
<b>4. Distribución de objetos sobre una superficie y definición de reglas</b>	<b>27</b>
4.1. Implementación del primer prototipo . . . . .	27
4.2. Ciclo de ejecución con <i>AIMA</i> . . . . .	31
4.3. Algoritmos de búsqueda de caminos para distribuir objetos . . . . .	34
4.3.1. Adaptación del algoritmo para múltiples soluciones . . . . .	37
4.4. Diseño del problema . . . . .	38
4.5. Redefinición de los métodos Valor y Coste del Camino . . . . .	42
4.6. Definición de reglas y sus efectos en la distribución . . . . .	43
4.6.1. Planteamiento original del sistema de reglas . . . . .	43
4.6.2. Reglas sobre las propiedades del objeto . . . . .	44
4.6.3. Reglas sobre la limitación de la distribución . . . . .	46
4.6.4. Implementación de las reglas . . . . .	48
<b>5. Integración de la herramienta en un programa 3D</b>	<b>50</b>
5.1. Implementación de operadores en <i>Blender</i> . . . . .	50
5.2. Utilidades y herramientas de la extensión . . . . .	51
5.3. Disposición de la interfaz gráfica de usuario . . . . .	55
5.4. Instalación de la herramienta en el programa 3D . . . . .	59

<b>6. Evaluación</b>	<b>62</b>
6.1. Método de evaluación . . . . .	62
6.2. Proceso de evaluación . . . . .	63
6.2.1. Primer experimento . . . . .	64
6.2.2. Segundo experimento . . . . .	70
<b>7. Discusión</b>	<b>79</b>
<b>8. Conclusiones y trabajo futuro</b>	<b>82</b>
8.1. Conclusiones del proyecto . . . . .	82
8.2. Trabajo Futuro . . . . .	83
<b>9. Contribución de los miembros</b>	<b>87</b>
9.1. Jose Daniel Rave Robayo . . . . .	87
9.2. Sergio José Alfonso Rojas . . . . .	89
9.3. Daniel Illanes Morillas . . . . .	90
<b>10. Introduction</b>	<b>93</b>
10.1. Motivation . . . . .	94
10.2. Initial Hypothesis . . . . .	95
10.3. Objectives . . . . .	96
10.4. Methodology . . . . .	97
10.4.1. Technologies and tools used . . . . .	97
10.4.2. Work plan . . . . .	98
10.5. Document structure . . . . .	98
<b>11. Conclusion and future work</b>	<b>101</b>
11.1. Project Conclusions . . . . .	101
11.2. Future Work . . . . .	102

<b>Bibliography</b>	<b>105</b>
<b>A. Notas de instalación del repositorio</b>	<b>111</b>
<b>B. Enlaces relacionados con el proyecto</b>	<b>116</b>
<b>C. Documentos del proceso de evaluación</b>	<b>117</b>
C.1. Cuestionario inicial del primer experimento . . . . .	117
C.2. Cuestionario inicial del segundo experimento . . . . .	120

*“Si no sabes a dónde quieres ir, no importa qué camino tomes”.*

- El gato de Cheshire.

Alicia en el país de las maravillas.

# Capítulo 1

## Introducción

El diseño de entornos digitales se ha convertido en una tarea crucial en la industria del entretenimiento. Desde el cine hasta los videojuegos, la creación de escenarios supone un trabajo exhaustivo y creativo, que conlleva invertir una cantidad considerable de tiempo para obtener un resultado satisfactorio.

En este sentido, las principales aplicaciones tecnológicas dedicadas a la creación de elementos 3D y su aspecto estético han evolucionado significativamente a lo largo de los años, incorporando herramientas cada vez más sofisticadas con el objetivo de hacer posible la construcción de espacios tridimensionales. De esta manera, permiten la generación topológica de objetos, así como la aplicación de texturas y materiales para crear distintos estilos que se adapten al proyecto. Sin embargo, crear un entorno digital no se basa principalmente en producir los elementos que componen la escena. Además de tratarse de una parte importante, la colocación o distribución de éstos en el espacio virtual es un factor determinante para lograr una buena composición que defina una experiencia inmersiva y coherente para el usuario.

Existen programas de pago como *Maya* o *Houdini*, e incluso de libre uso como *Blender*, que incorporan la posibilidad de repartir varios objetos sobre una superficie. No obstante, su funcionalidad es limitada, pues no proporcionan libertad al usuario de diseñar el entorno a su gusto. Sumado a ello, cabe destacar que su complejidad y falta de intuitividad pueden resultar un obstáculo significativo, especialmente para aquellos usuarios que carecen de experiencia previa en el uso de este tipo de programas.

Estos programas poseen una gran variedad de extensiones que añaden funcionalidades

adicionales y, mediante ellas, dotan al usuario de una serie de facilidades que no tendrían con el programa base. Con ello, se han ido desarrollando muchas herramientas nuevas para artistas y diseñadores que pretenden apoyar la creación tanto de modelos individuales como escenarios complejos.

En consecuencia, se ha decidido desarrollar una extensión que permita la distribución de uno o más objetos sobre una superficie dada, apoyándose en diferentes algoritmos parametrizados y manipulables por el usuario, con el fin de inhibir lo mínimo su creatividad. Para el uso de múltiples algoritmos, se utilizará *AIMA*, una implementación del libro universitario “Inteligencia Artificial: Un Enfoque Moderno”, el cual recopila una gran variedad de técnicas y algoritmos relacionados con la inteligencia artificial, y de los cuales se hará uso en este documento.

Además, se desarrollará de manera que pueda ser extrapolado a otros programas 3D, manteniendo la funcionalidad principal totalmente ajena al programa y su interfaz de programación de aplicaciones (comúnmente conocida por su acrónimo en inglés *API*).

Este documento tratará sobre el desarrollo de una herramienta para *Blender*. Se detallará desde la idea que motivó su realización y los obstáculos que se presentaron, hasta los nuevos conocimientos y herramientas que se emplearon para llevar el proyecto a cabo.

## 1.1. Motivación

El desarrollo de este trabajo específico se apoya en la necesidad de herramientas que faciliten tareas de diseñadores y artistas. Esto les permitiría aligerar su carga de trabajo y ahorrar tiempo en la creación de escenarios. Además supondría un aumento en la eficiencia y una mejora en la calidad de sus obras.

Así, implementar extensiones para programas 3D se ha vuelto una práctica cada vez más habitual y, en muchos casos, necesaria, ya que la inclusión de nuevas funcionalidades resulta enriquecedora y permite mejorar sustancialmente el contenido del programa. Además, este incentivo se ve reforzado cuando se facilita el acceso a la *API*, lo que permite que la comunidad interactúe con ella y colabore en su evolución, según su definición en la Sección 1.4.

Por ello, con el fin de diseñar una solución que satisfaga plenamente las expectativas del

mercado, se ha optado por desarrollar un prototipo empleando el programa *Blender* (véase Sección 1.4.1), que permita experimentar y diseñar una solución óptima a partir de la cual se puedan obtener resultados satisfactorios.

La solución propuesta se cree que solventará el problema expuesto, ya que el artista o diseñador podrá centrarse más en términos de composición y en la disposición de grupos de objetos, en vez de individualmente, y con ello potenciar la velocidad de producción. Asimismo, si bien la herramienta puede ser útil para contextos de proyectos grandes, en los que la creación del escenario es un componente primordial, también puede ser utilizada en proyectos que requieran un desarrollo más improvisado y efímero, como es el caso de las *Game Jams*<sup>1</sup>.

Para finalizar, el propósito de este trabajo consiste en definir un estilo de implementación que sea escalable; es decir, que permita continuar su desarrollo según se vayan planteando posibles ampliaciones. Esto permite llegar a unas cotas claras que cumplan los objetivos planteados, pero que, a su vez, dejen espacio para que cualquier usuario o desarrollador pueda ampliar las funcionalidades.

## 1.2. Hipótesis de partida

Según lo expuesto en la sección anterior, el objetivo principal es realizar una herramienta que, a la hora de crear entornos 3D, ayude a los usuarios en términos de tiempo y composición. Por tal motivo, se ha planteado la siguiente hipótesis que guiará el rumbo del proyecto:

**Hipótesis 1 (H<sub>1</sub>).** *Utilizando la herramienta de distribución automática de objetos sobre superficies desarrollada, un usuario es capaz de crear escenarios tridimensionales empleando menos tiempo que sin usar dicha herramienta.*

Validar H<sub>1</sub> proporciona información sobre si la herramienta cumple con lo estipulado en este proyecto. Sin embargo, dicha validación no da información sobre si los usuarios están lo suficientemente satisfechos con el resultado, como para preferir usar la herramienta en vez de realizarlo manualmente. Por conseciente, surge la siguiente hipótesis:

---

<sup>1</sup>Una *Game Jam* es un evento que reúne a desarrolladores con el objetivo de crear uno o varios videojuegos en un lapso corto de tiempo, generalmente de entre 24 y 48 horas.

**Hipótesis 2 (H<sub>2</sub>).** *La mayoría de los usuarios encuentran que la calidad de los resultados obtenidos por la herramienta es mejor o igual, en comparación a la calidad de los resultados que se pueden conseguir a mano, valorando la precisión y flexibilidad del proceso.*

### 1.3. Objetivos

Como se ha mencionado previamente, el trabajo pretende desarrollar una herramienta de distribución de objetos en un espacio tridimensional. Teniendo esto y la hipótesis previamente planteada en consideración, los objetivos que se plantean son los siguientes:

- **Desarrollar un primer prototipo que cumpla los requisitos mínimos.** Se creará un programa prototipo que sea capaz de posicionar objetos sobre una superficie dada. Además, dotar de personalización y flexibilidad a la propia distribución.
- **Establecer un sistema de reglas y restricciones que tendrán que seguir los objetos a distribuir.** Se definirá qué parámetros referentes a cada tipo de objeto son de mayor importancia para el usuario y cómo se podrán modificar para limitar la capacidad de distribución de los mismos.
- **Implementar estos sistemas mediante diversos algoritmos de distribución, de manera que se alcance el resultado final más óptimo en cuanto a solución y eficiencia.** Se iterará sobre diferentes algoritmos, tratando de aclarar cual, o cuales, generan una resolución de la distribución con mayor utilidad para el artista, y mayor versatilidad con relación a la modificación de posición de los objetos.
- **Implementar una serie de utilidades para un programa específico en la extensión que permitan al usuario acceder con mayor facilidad a las funcionalidades principales del mismo.** Se aportarán herramientas que faciliten en qué partes de la superficie se pueden colocar objetos, que apliquen subdivisiones, que guarden perfiles de distribución y otras posibles funcionalidades que se puedan diseñar a lo largo del desarrollo.
- **Diseñar un sistema que distribuya objetos sobre una superficie en cualquier programa de modelado 3D sin distinción.** Se establecerán una serie de clases que puedan ser accedidas desde diferentes aplicaciones sin exclusividad de una concreta. Se planteará, para ello, un programa de modelado específico sobre el que aplicar la

solución, que teóricamente se podría extraer a otros programas.

- **Implementar una interfaz de usuario en el programa de modelado específico, que permita utilizar este sistema desde una jerarquía de paneles.** Se dividirán por paneles las diferentes secciones de la extensión. De esta forma, se permitirá al usuario interactuar intuitivamente con cada uno de ellos.
- **Evaluar las interacciones de un grupo de usuarios con la extensión y realizar los ajustes pertinentes una vez analizados los datos que se recopilen.** Se tendrá en cuenta el perfil de cada usuario, de manera que se seleccionará qué elementos de la retroalimentación se tendrán más en cuenta.
- **Establecer una conclusión con los datos recopilados** Se determinará si la hipótesis planteada originalmente se cumple. Además, se planteará, a partir de esta conclusión, qué elementos del trabajo se podrán ampliar para alcanzar un mejor resultado.

## 1.4. Metodología

En este apartado se comentarán las herramientas y programas utilizados para la resolución de este proyecto, así también como la organización de los miembros para llevar a cabo las distintas tareas que conforman el trabajo.

### 1.4.1. Tecnologías y herramientas usadas

Para llevar a cabo la implementación de la extensión se necesita, en primer lugar, un programa de modelado 3D sobre el que desarrollar el trabajo. Un programa de modelado 3D es una herramienta digital utilizada para crear modelos tridimensionales de objetos, escenarios o personajes en un entorno virtual (véase Sección 2.2). Suele ser usado por profesionales con un perfil artístico, aunque no se excluyen aquellos técnicos, pues también permiten la realización de simulaciones físicas. Así, para este proyecto, se ha elegido *Blender* debido a su versatilidad y sencillez de uso. Además, una de sus características que más destacan es que es de código abierto, es decir, su código fuente es libre uso. Esto facilita la creación de extensiones y herramientas que amplíen sus funcionalidades. Adicionalmente, cuenta con una *API* (Interfaz de Programación de Aplicaciones) extensa y bien documentada. En términos generales, una *API* tiene como objetivo permitir la comunicación entre elementos externos

y el contenido interno del programa, accediendo así a datos o acciones específicas del mismo.

Para programar una extensión enfocada en *Blender*, se hace uso del lenguaje interpretado *Python*. Si bien esto puede parecer una ventaja, debido a su sencillez semántica y claridad visual, posteriormente se señalarán las desventajas de este lenguaje, tales como la falta de tipado y su limitada eficiencia.

Este fue el lenguaje empleado para el desarrollo de la extensión. Si bien se pudo haber utilizado cualquier editor de texto, se decidió usar el entorno *Visual Studio Code (VS Code)*. Éste se trata de un entorno de desarrollo integrado (comúnmente conocido como *IDE* por sus siglas en inglés), una aplicación informática que implementa funcionalidades para facilitar el desarrollo de aplicaciones o herramientas y que además incorpora módulos para trabajar sobre distintos lenguajes de programación o incluso varias plataformas. Uno de los motivos principales por los que se eligió este *IDE* es debido su sencillez e interfaz gráfica elegante. Sin embargo, el motivo principal por el que no se usó otro entorno está relacionado con la existencia de una extensión para *VS Code*, que facilita el desarrollo de herramientas para *Blender*. Esta extensión tiene de nombre “Desarrollo en *Blender* dentro de *VS Code*” (originalmente como *Blender Development in VS Code*<sup>2</sup>). Asimismo, no sólo aporta atajos de teclado para la creación de operadores (véase Sección 5.1), sino que también incorpora la posibilidad de depurar el código, enlazando el propio programa *Blender* con el entorno *VS Code*, lo que agiliza la detección de errores y el desarrollo.

Por otro lado, como se menciona en la Introducción, éste trabajo usa la biblioteca *AIMA*, la cual es una implementación del libro “*AIMA. Inteligencia Artificial: Un Enfoque Moderno*”[1] (véase Sección 2.2.2). Ésta es, con seguridad, la herramienta más importante del proyecto, pues incorpora gran parte de las funcionalidades y algoritmos utilizados en el trabajo, que permiten la distribución de objetos sobre superficies. En la Sección 4.3 se explica qué algoritmos fueron utilizados, así como otros elementos integrados en ésta biblioteca.

Por último, para la organización del proyecto, se hizo uso de distintas aplicaciones que facilitan la planificación de las tareas a futuro, con el objetivo de llevarlas a cabo de manera eficiente. En principio, para mantener un orden y copias de seguridad del desarrollo, se utilizó el sistema de control de versiones *Github*, en el cual se encuentra alojado el repositorio de este estudio (véase Apéndice Enlaces relacionados con el proyecto). En relación a la creación de tareas, se utilizó *Pivotal Tracker* para monitorizar el desarrollo de éstas. Principalmente

---

<sup>2</sup>Repositorio a la extensión auxiliar: [https://github.com/JacquesLucke/blender\\_vscode](https://github.com/JacquesLucke/blender_vscode)

la duración de las tareas tenían un plazo de dos semanas, coincidiendo así con el tiempo entre reuniones acordado con los tutores del trabajo.

#### 1.4.2. Plan de trabajo

Para realizar un seguimiento del progreso del proyecto y crear una rutina de trabajo, se ha acordado con los tutores tener reuniones periódicamente. Toda la planificación se encuentra en la Tabla 1.1.

Fecha	Trabajo Realizado
Mayo 2022	Planteamiento de la idea original, formación de grupo de trabajo y acuerdo con directores de proyecto.
Septiembre 2022	Organización del trabajo a intervalos regulares, elección de tecnologías a usar, planificación periódica de sesiones con directores e inicio del trabajo.
Octubre 2022	Diseño y creación del primer prototipo de distribución. Primer estudio sobre el estado del arte.
Noviembre 2022	Implementación de biblioteca <i>AIMA</i> y primeros algoritmos. Diseño e implementación de primeras utilidades.
Enero 2023	Diseño e implementación del primer sistema de reglas y optimización de algoritmos utilizados.
Febrero 2023	Utilidades de pintado de vértices, versión avanzada de sistema de reglas y configuración final de la interfaz de usuario.
Marzo 2023	Sistema final de reglas, adición de últimos algoritmos utilizados para la distribución y utilidades finales.
Abril 2023	Conexión y coordinación de todos los sistemas y comienzo de la memoria.
Mayo 2023	Compleción de la memoria, evaluación y pruebas con usuarios, y obtención de conclusiones finales.
Junio 2023	Presentación del trabajo ante el tribunal y publicación del mismo.

**Tabla 1.1:** *Plan de Trabajo.*

### 1.5. Estructura del documento

En esta sección se describirá brevemente cada uno de los capítulos y secciones posteriores que componen el documento, con el fin de proporcionar una visión general del contenido y del orden en que se presentarán los aspectos fundamentales de la investigación.

En el Capítulo 2 posterior a este, se estudiará el estado del arte. Se desarrollará sobre qué tecnologías juegan un papel similar a lo que se plantea en este trabajo y se dejará claro qué características se dejan en el tintero, de manera que se muestre por qué derroteros se llevará la extensión. Además de las tecnologías y aplicaciones similares, se hablará sobre los algoritmos que componen la estructura interna del trabajo y su situación en la actualidad.

La arquitectura del proyecto y su organización se encuentra detallada en el Capítulo 3. Se describen los elementos que componen el programa y cómo estos son envueltos en tipos y clases. Además se argumenta y analiza dicha estructura junto con sus dependencias y limitaciones.

En el Capítulo 4 se hablará sobre la elección de algoritmos de distribución, su funcionamiento y los beneficios que se han encontrado en los algoritmos finales respecto a aquellos con los que se comenzó el trabajo. Además, se desarrollará la implementación de la biblioteca *AIMA* [1] para *Python*. Por otro lado, se tratarán las reglas y restricciones. Estas se combinan con los algoritmos de distribución explicados en la Sección 4.3 para definir cómo se van a distribuir los objetos por cada superficie.

El Capítulo 5 será el último capítulo de contribución, en el que se hablará del flujo de trabajo que se debe llevar al usar la extensión. Aquí se explicarán los diferentes paneles que tiene la interfaz de usuario y de qué manera el usuario puede aprovechar todas las utilidades que se le brindan.

La evaluación con usuarios se desglosará en el Capítulo 6, donde se hablará del método de evaluación. Sobre este método se completará la información que defina el proceso de evaluación, incluyendo los experimentos realizados, el número de usuarios que han participado y los perfiles que tienen. Esta evaluación permitirá conseguir una serie de resultados que, una vez recolectados, dejarán paso a una sección sobre análisis de los mismos.

El Capítulo 7 desarrollará las ventajas e inconvenientes que presenta el trabajo. Se comparará con todo aquello que se contempla en el Capítulo 2, y se determinará qué objetivos se han completado y cuáles no se han sobrepasado.

Como síntesis, el Capítulo 8 dará paso a la conclusión final, a la que se ha llegado después de los resultados obtenidos. Asimismo, se hablará del trabajo que ha quedado pendiente por desarrollar en el futuro.

Por último, el capítulo 9 contendrá las contribuciones aportadas por los miembros de este trabajo.

# Capítulo 2

## Estado del arte

El trabajo que se plasma en este documento no proviene de una idea nueva. Se relaciona con un ámbito probabilístico conocido como la distribución de probabilidad. Además, se abarcan conceptos paralelos como la búsqueda de caminos y la inteligencia artificial.

Cabe destacar que para lograr desarrollar esta extensión ha habido que contemplar otros proyectos ya desarrollados en este campo. La realización de este trabajo no habría sido posible sin ellos. A esto además se le debe sumar el estudio y comprensión de los diferentes conceptos relacionados con esta tecnología.

Por lo tanto, en este capítulo se hablará del estado del arte de algunas definiciones, abarcando desde la generación procedural hasta diferentes algoritmos de búsqueda. Además, se resaltarán y analizarán algunos proyectos que ofrecen una funcionalidad similar a este trabajo y que han podido servir de inspiración.

### 2.1. Generación procedural

La generación procedural de contenidos es el proceso de diseño de recursos mediante algoritmos informáticos en lugar de aplicar directamente recursos humanos. Refiriéndose por contenido a los recursos de un videojuego, que son la parte de los datos del juego que no es código informático. Esto incluye a texturas 2D, modelos 3D, entornos, animaciones, efectos de sonido y música, entre otros [2].

Los primeros usos de generación procedural que se conocen datan de 1976, imple-

mentada en una expansión del juego de rol *Dungeons & Dragons* en la cuál se permitía al *Dungeon Master* (el jugador encargado de organizar y dirigir el curso de la partida) generar las mazmorras y el terreno mediante el uso de tiradas de dados. Esto se limitaba a un entorno físico pero sentaría precedentes [3] [4].

Este concepto se puede encontrar llevado a los videojuegos en títulos como *Beneath Apple Manor* (1978) o *Rogue* (1980). Los sistemas de generación procedural en estos juegos se encargarían de producir escenarios o entornos de gráficos sencillos, pero con suficiente complejidad como para suponer un desafío a los jugadores [5]. Estos juegos marcarían un hito en el uso de la generación procedural en los videojuegos, que evolucionaría con el paso del tiempo. En la actualidad podemos encontrar infinidad de ejemplos de su uso en diferentes títulos como puede ser: *Crusader Kings II* (2012), *Shadow of Mordor* (2014), *Slay the Spire* (2019), *Rogue Legacy 2* (2020), entre otros [6]. El uso de esta tecnología se ha expandido a muchos ámbitos del desarrollo, al punto de poder llegar a generar niveles, texturas, personajes, armas, vegetación, o estructuras [7].

Lo que caracteriza principalmente a esta técnica de producción es que debe ser rápida, con la capacidad de generar contenido en ordenadores actuales. Además, debe ser aleatoria y estar estructurada de tal manera que cree contenido que se adhiera a las expectativas del desarrollo. Y por último, ser ajustable de manera sencilla para facilitar precisamente la velocidad y variabilidad necesarias para ser interesante [8].

No obstante, no se limita solamente al medio de los videojuegos. Tal es el caso con *Pixar* y su motor de renderizado 3D, *RenderMan*. Este permitía generar materiales y texturas mediante algoritmos, así como crear modelos sencillos por código [9][10]. Otro ejemplo de su uso se encuentra dentro del cine y la televisión, como sucede con *MASSIVE (Multiple Agent Simulation System in Virtual Environment)*. *MASSIVE* es un paquete de software diseñado para efectos visuales. Su principal característica es la de generar grupos de agentes, por ejemplo personajes, que pueden actuar con comportamientos y acciones únicos. Esto permite generar grandes y variados grupos o multitudes “realistas” en todo tipo de filmes y metrajes. [11] [12].

La generación procedural ha demostrado ser una rama importante en la creación de contenidos y recursos, aunque aún se requiera de más investigación y evolución [13]. Más adelante se hablará de los elementos que actualmente fundamentan la generación procedural, como pueden ser los números pseudo-aleatorios (véase Sección 2.1.1) o los diferentes

algoritmos que los emplean (véase Sección 2.2.2).

### 2.1.1. Números pseudo-aleatorios

Muchos algoritmos dependen de muestras aleatorias para funcionar correctamente, sin embargo, un ordenador no puede generar números aleatorios [14]. Los “generadores de números aleatorios” que se usan son de hecho funciones deterministas que producen una secuencia periódica de números. Estos se basan en un valor inicial o *semilla* para empezar a generarla, pretendiendo que parezca y se comporte como una secuencia de números aleatorios genuina. Por esta razón, se les conoce como “generadores de números pseudo-aleatorios” [15].

Una desventaja de este tipo de generadores es su incapacidad de generar infinitos números “aleatorios”. Eventualmente alcanzan un estado previamente visitado que desemboca en la repetición de la secuencia. No obstante, una cualidad clave de estas funciones es su capacidad de repetir exactamente la misma secuencia de números sin almacenarlos previamente. Esto es una característica esencial para análisis o depuración de código, por ejemplo, pues contando con una semilla inicial se pueden recrear procesos “aleatorios” una y otra vez [16].

Los números pseudo-aleatorios se emplean en multitud de situaciones dentro del sector de la informática. En lo referente a este proyecto, como se ha mencionado previamente, se utilizan en los diferentes algoritmos que sirven para la distribución de objetos.

### 2.1.2. Búsqueda de caminos

La búsqueda de caminos, o *pathfinding* en inglés, es una de las posibles aplicaciones de la inteligencia artificial [17]. Esta rama deriva del problema del camino más corto, perteneciente a la teoría de grafos. Un grafo es un estructura abstracta, asociada a las matemáticas y constituida por vértices y aristas. Las aristas se encargan de conectar cada par de vértices entre sí. Cada vértice puede representar cualquier elemento que requiera el contexto, ya sea una posición en un escenario, una ciudad en un mapa o un estado de un personaje, entre muchos otros. Por otro lado, dependiendo de si se pueden recorrer las aristas en un sentido o en ambos, podemos estar refiriéndonos a grafos dirigidos o no dirigidos, respectivamente. Además, a la longitud de las aristas se les suele referir como “pesos”. Estos “pesos” se utilizan normalmente para referirse al coste de atravesar dicha arista. Se usan con frecuencia en algoritmos de búsqueda [18] [19].

La búsqueda de caminos generalmente se refiere a encontrar la ruta que recorre menos

vértices desde uno inicial hasta un vértice destino. Aunque si bien esto puede ser lo más común, no se limita sólo a encontrar el camino más corto. También se puede referir a otro tipo de criterios. Entre ellos podemos encontrar: cuál es el más rápido, el que tiene menor coste, aquel que cumpla ciertas condiciones, etc. Estas normas están directamente relacionadas con el algoritmo de búsqueda y el tipo de grafo empleados [20].

Se pueden encontrar multitud de implementaciones de búsqueda de caminos en distintas áreas tecnológicas como son: la radio-navegación (*GPS*), los videojuegos, la robótica o la logística, entre otros. Además, ésta puede ser aplicada en entornos estáticos, dinámicos y de tiempo real [21]. Para casos como la radio-navegación, el objetivo suele ser encontrar el camino más rápido hacia el destino, aunque eso conlleve no elegir el camino más corto. En cambio, para el caso de la robótica o los videojuegos, es frecuente que el objetivo sea evitar obstáculos y encontrar el camino más práctico en diferentes terrenos o con ciertas restricciones[22].

Como ya se ha mencionado, la búsqueda de caminos está fuertemente relacionada con la teoría de grafos. Más adelante se desarrolla con mayor detalle la relación de este trabajo con la misma (véase Sección 4.2), así como con los distintos algoritmos de búsqueda que se emplean (véase Secciones 2.2.2 y 4.3).

## 2.2. Tecnologías relacionadas

### 2.2.1. Modelos 3D y programas de modelado 3D

Un modelo 3D es, por lo general, un conjunto de vértices y aristas, conectados mediante polígonos. A estos vértices se les otorga una posición, escala y orientación arbitrarias en un entorno tridimensional simulado. Combinando estos puntos por sus aristas a través de formas poligonales, como triángulos o cuadriláteros, se constituye lo que se conoce como la superficie del objeto. Este conjunto es comúnmente conocido como “malla” y permite obtener una representación visual [23] [24] [25]. Pueden ser producidos manualmente por un artista, escaneados mediante cámaras a través de programas especializados o, como se ha mencionado anteriormente y siguiendo el enfoque de este trabajo, mediante el uso de generación procedimental. Los modelos 3D se utilizan hoy día en diversos medios, como videojuegos, películas, arquitectura, ilustración, ingeniería y publicidad comercial [23].

En un inicio se dependía de papel y lápiz para diseñar un producto. Más tarde aparece-

rían programas de diseño asistido por ordenador o *CAD* (del inglés, *computer-aided design*), aún limitados al diseño en dos dimensiones [26]. En la actualidad, para dar forma a estas ideas se emplean programas o aplicaciones de modelado 3D. Estos permiten representar los modelos en un entorno tridimensional. Al tratarse de una representación en tiempo real, también facilita a sus usuarios la modificación y personalización de los mismos mediante el uso de múltiples herramientas. Entre los más conocidos se encuentran *AutoCAD*, *Maya*, *SketchUp*, *ZBrush* o *Blender* [27]. Este último guarda una estrecha relación con el trabajo realizado y se profundiza en sus peculiaridades más adelante (véase Sección 5).

### 2.2.2. *AIMA*. Inteligencia Artificial: Un Enfoque Moderno

La inteligencia artificial o *IA* es realmente un campo muy extenso, pues ésta puede abarcar desde sencillos programas con funcionalidades básicas hasta complejos y profundos sistemas de aprendizaje o reconocimiento. A lo largo de la historia ha sido descrita de diferentes formas, en palabras de R.C. *Schank*, “[...], uno de los problemas de definir la inteligencia artificial [...] es que ésta [...] puede ser prácticamente cualquier cosa.” [28], o como describe *P. Wang*: “Los seres humanos se distinguen de los animales y las máquinas principalmente por su habilidad mental o cognitiva, comúnmente llamada ‘inteligencia’, y la *IA* es el intento de reproducir dicha habilidad en un sistema informático” [29].

El libro “Inteligencia Artificial: Un Enfoque Moderno” (o *AIMA*), publicado inicialmente en 1995, a fecha de hoy continua actualizándose con nuevos algoritmos y lenguajes de programación. Es un escrito considerado como estándar y referente en el campo de la inteligencia artificial, que además define la misma como el conjunto de “agentes que reciben impresiones del entorno y ejecutan acciones” [1].

Su contenido está pensado para ser impartido e implementa distintos programas sobre inteligencia artificial en pseudo-código, sumado a explicaciones de los mismos. Además, para complementar el libro, existe un repositorio con todas las implementaciones en diferentes lenguajes de programación, entre ellos *Python*, y en la cuál se basa el funcionamiento de los distintos algoritmos de este trabajo.<sup>1</sup>

A continuación se hablará sobre distintos algoritmos y heurísticas implementados en el entorno de *AIMA* y que tienen relación o han sido incluidos en el trabajo realizado.

---

<sup>1</sup><https://github.com/aimacode/aima-python>

## **Algoritmo Breadth First Search**

*Breadth First Search* o Búsqueda en Anchura es un algoritmo de búsqueda ciego, es decir, que realiza su recorrido sin tener en cuenta si está realizando el mejor camino. Consiste en explorar desde la raíz todos los nodos conexos a la misma, y posteriormente todos los conexos a dichos nodos. Se finaliza repitiendo esta acción hasta que se recorren todos los nodos del árbol [30].

## **Algoritmo Djikstra**

El algoritmo Dijkstra soluciona el problema de encontrar el camino más corto desde cualquier punto de un grafo a otro [31]. Se basa en que cualquier subcamino  $B \rightarrow D$  del camino más corto  $A \rightarrow D$  entre los vértices A y D es también el camino más corto entre los vértices B y D. Djikstra utiliza esta propiedad en el sentido opuesto, es decir, se sobreestima la distancia a cada vértice desde el vértice inicial. Entonces se visita cada nodo y sus vecinos para encontrar el camino más corto entre ellos [32].

## **Algoritmo A\***

El algoritmo  $A^*$  (o A-estrella) resuelve el mismo problema que Djikstra. La diferencia principal con Djikstra es que introduce heurísticas [33]. Una heurística puede referirse a cualquier estrategia utilizada en resolución de problemas, ya sea un programa, una estructura de datos, un proverbio o un conocimiento. Pero no sirve cualquier estrategia, tiene que tener una cualidad muy específica: que sea útil, pero que no garantice el éxito absoluto [34]. De esta manera, empleando una heurística,  $A^*$  busca en cada iteración el camino que probablemente se aproxime más al resultado que se está buscando [35].

## **Algoritmo Simulated Annealing**

El algoritmo *Simulated Annealing*, o en español *Temple Simulado*, es una heurística probabilística, es decir, se basa en números aleatorios (véase Sección 2.1.1) para influenciar las elecciones que toma durante su ejecución [36]. Se le considera a su vez un algoritmo de “búsqueda local”, ya que dada su naturaleza es posible que nunca explore una región del espacio de búsqueda en la que exista una solución. A cambio, este tipo de algoritmos consumen muy poca memoria, y encuentran soluciones razonables en espacios de búsqueda posiblemente infinitos [33].

Debe su nombre precisamente al templado de metales. Esto se refiere a una técnica metalúrgica que consiste en elevar la temperatura de un metal al punto en el que sus partículas comienzan a ordenarse aleatoriamente en su fase líquida, para posteriormente reducir la temperatura y así enfriarlo lentamente [37] [38]. Las bases del algoritmo surgen de relacionar este proceso con la necesidad de resolver extensos problemas de combinatoria.

En su implementación, la heurística *Simulated Annealing* comienza con una solución y continuamente busca en sus vecinos por otra solución con un coste menor. Si la encuentra, entonces la reemplaza. Este proceso continua hasta que no se consiguen encontrar mejores estados que el actual. A esta situación final se le conoce como “estado óptimo local” [39][40].

### Algoritmo Hill-Climbing

El algoritmo *Hill-Climbing*, o en español *Escalada Simple*, se asemeja al algoritmo previamente mencionado, *Simulated Annealing*, en que también es una heurística de “búsqueda local” [41].

En su implementación, el algoritmo comienza con una solución arbitraria inicial y en cada iteración avanza hacia soluciones cercanas con más valor, es decir, que se aproximan más a una solución óptima. Termina cuando alcanza un “pico” donde no es capaz de encontrar ningún estado cercano con más valor. Como se ha explicado en el apartado anterior (véase Sección 2.2.2), al tratarse de un algoritmo de “búsqueda local”, *Hill-Climbing* no ve más allá de sus estados más inmediatos, por lo que la solución puede no ser la mejor a escala global, pero sí lo será dentro de su espacio de búsqueda [1] [42].

Uno de los problemas que se pueden encontrar al utilizar *Hill-Climbing* son los *Plateau* o *Mesetas*, que consisten en áreas planas en el espacio de búsqueda en los que el valor de todos los estados cercanos es muy similar entre sí y no existen posibles mejores soluciones al alcance. Esto puede provocar que el algoritmo deje de progresar [43]. No obstante, este problema se puede evitar limitando la cantidad de iteraciones que se le permite realizar en un estado de bajo progreso, sorteando así un bucle infinito de búsqueda, como se lleva a cabo en la integración de la heurística en este trabajo (véase Sección 4.3).

## 2.3. Revisión de programas y extensiones relacionados con la distribución de objetos

Programas de código abierto o con una *API* (del inglés *Application Programming Interface*; se refiere a las reglas y protocolos para conectar, integrarse y extender un programa [44]) bien documentada, sumados a la heterogeneidad de internet, pueden llegar a producir herramientas muy potentes y versátiles. Esto se refiere a casos como el de este trabajo. Aunque como se menciona anteriormente, ha sido necesario observar proyectos ya realizados para poder llevarlo a cabo (véase Sección 2). A continuación se hará un breve análisis sobre alguna de estas herramientas, así como del entorno en el que se ejecutan y el estado de los mismos.

En primer lugar, *Blender* ya cuenta con una herramienta propia de distribución de objetos sobre superficies<sup>2</sup>. Inicialmente se encuentra desactivada, y es necesario investigar o conocer de ella para encontrarla dentro del propio programa. Sus opciones son bastante limitadas y la interfaz no se encuentra en un lugar intuitivo o fácilmente accesible. Aunque, como punto a favor a la hora de distribuir, genera instancias de malla en lugar de duplicados, es decir, genera referencias que permiten reflejar los cambios en las copias al modificar el original. Esta es una elección muy práctica y conveniente para los artistas, que además tiene como cualidad que consume muy poca memoria durante su uso.

Por otro lado existe *BagaPie*<sup>3</sup>, una extensión gratuita y muy popular dentro de la comunidad. Es una herramienta difícil de dominar debido principalmente a sus interfaces poco intuitivas, pero lo compensa con su versatilidad. Además de ser muy eficiente, porque se apoya en una característica de *Blender* conocida como *geometry nodes* o nodos de geometría. Este es un sistema basado en gráficos de nodos que permite crear y manipular geometría de forma procedural y controlada dentro de *Blender* (véase Sección 2.1). Sin embargo, esto hace que la extensión deba limitarse únicamente al entorno de *Blender*, donde opera dicho sistema de nodos. Además, *BagaPie* no se limita únicamente a la distribución de objetos. También ofrece otras funcionalidades como la generación procedural de una serie de objetos o un mejorado sistema de matrices para ordenar objetos, entre otros. Por lo tanto, aunque ofrece varias opciones a la hora de distribuir, el nivel de personalización es muy reducido.

---

<sup>2</sup><https://blender-addons.org/object-scatter-addon/>

<sup>3</sup><https://abaga.gumroad.com/l/BbGVh>

Existen otras extensiones elaboradas por y para usuarios que se emplean tanto a nivel principiante como profesional dentro del mercado de *Blender*<sup>4</sup>. Sin embargo, muchas basan su funcionamiento en nodos de geometría o sistemas de partículas, dificultando su extrapolación a otros programas. Otras aportan implementaciones muy limitadas y solo permiten personalizar de formas muy sencillas la distribución. Además, muchas de ellas cuentan con curvas de aprendizaje muy pronunciada, normalmente agravada por la falta de documentación explícita. Por último, también existen proyectos que fueron abandonados a mitad de desarrollo, que han ido quedando obsoletos con el paso de actualizaciones del programa, o que son difícilmente accesibles por el usuario medio debido al elevado precio de los mismos.

También se han contemplado otras herramientas pertenecientes a programas externos a *Blender*, como son el caso de *Maya* y *3dsMax*. En el caso de *Maya*, un programa de modelado y animación 3D, siempre han existido métodos indirectos para lograr distribuir objetos sobre superficies, por ejemplo mediante *scripts* basados en sistemas de partículas. No obstante, esta forma de distribuir era muy limitada y en muchos casos dificultaba la manipulación posterior del resultado producido. No es hasta recientemente que surgió *EnvIt*<sup>5</sup>, una herramienta específica y muy potente para la distribución. Aunque continúa en desarrollo, aún se encuentra limitada al entorno de *Maya*.

Por otro lado se encuentra *3dsMax*, otro programa de modelado y animación 3D. Este cuenta con su propia herramienta, la cuál goza de variedad de opciones y alternativas para modificar y adaptar la propia distribución. Sin embargo, cuenta con escasa documentación en la que apoyarse<sup>6</sup>. Además, al tratarse también de *software* propietario o de código cerrado, y ser un programa de pago por suscripción, su uso se ve limitado en cuanto accesibilidad y transparencia en las operaciones. Por último, cabe también mencionar su mercado de extensiones, que incluye herramientas similares de distribución. Aunque éstas son bastante escasas y poco populares entre los usuarios de *3dsMax*.

Estas herramientas no solo se limitan a programas de modelado. Motores de gráficos 3D de tiempo real gratuitos como *Unreal Engine* o *Unity* también cuentan con paquetes integrados de extensiones que permiten colocar objetos sobre mallas de manera controlada. Respecto a *Unreal*, la más conocida es su herramienta *Foliage*, o *Follaje* en español, que permite entre otras cosas personalizar la reglas que seguirán cada uno de los objetos al

---

<sup>4</sup><https://www.blendermarket.com/search?>

<sup>5</sup><https://www.envitscript.com/>

<sup>6</sup><https://help.autodesk.com/view/3DSMAX/>

momento de ser distribuidos individualmente, aspecto similar al que se da en este trabajo. Esto incluye una amplia variedad de propiedades generales y específicas, desde variación en rotaciones y posiciones hasta personalización de materiales, sombras y mallas. Además, se apoya en multitud de micro-herramientas como pinceles o instrumentos de selección para facilitar su uso [45]. *Unreal* además cuenta también con un mercado de recursos y extensiones<sup>7</sup> donde se encuentran multitud de otras herramientas de usuarios dedicadas a este y otros propósitos. El caso de *Unity* difiere significativamente del de *Unreal*, al menos en su herramienta predeterminada. El paquete integrado por defecto llamado *Polybrush*<sup>8</sup> consta de una herramienta de distribución mucho más simple, con funcionalidades básicas y mínima personalización. Se limita en sus funcionalidades a pinceles sencillos y escasas opciones de adaptación. No obstante, su mercado de recursos<sup>9</sup> se asemeja en contenido al de *Unreal*, llegando incluso a superarle en cantidad y variedad.

En resumen, las herramientas de distribución son un mercado que aún se encuentra en expansión y evolución, aumentando su diversidad y calidad día a día. No obstante, existe un riesgo generalizado de saturación generado por extensiones de baja calidad y con poca o nula diferenciación entre sí. Esto puede llegar a perjudicar a la larga a los usuarios que pretendan emplear estas herramientas como apoyo en el desarrollo de sus proyectos. Es precisamente por razones como esta que la incorporación de nuevas extensiones de calidad, accesibles y personalizables, además de código abierto y gratuitas, sea un paso adelante en pos del desarrollo de este campo y un beneficio para todos los usuarios de estas aplicaciones.

---

<sup>7</sup><https://www.unrealengine.com/marketplace/en-US/store>

<sup>8</sup><https://unity.com/features/polybrush>

<sup>9</sup><https://assetstore.unity.com/>

# Capítulo 3

## Arquitectura del proyecto

En este capítulo se detallarán qué ficheros forman parte del proyecto y cómo estos se relacionan entre sí. Además de indicar cuáles son los módulos que conforman el proyecto y qué ficheros forman parte de ellos.

En el repositorio <sup>1</sup> (Apéndice B), se encuentra una estructura de carpetas jerárquicas, en las cuales se almacena el código fuente del funcionamiento de la extensión. Como bien se comentó anteriormente, se utilizan ciertas clases de *AIMA* (véase Sección 2.2.2) para complementar la estructura de los algoritmos de búsqueda en el proyecto. Por otro lado, existen clases envolventes cuyo principal objetivo es abstraer información de elementos tridimensionales. Esto se realiza con el propósito de independizar la lógica del programa al que está asociada la herramienta.

Cabe volver a recalcar que el proyecto se realizó usando el *IDE Visual Studio Code* (ir a la Sección 1.4.1). En dicho entorno, se empleó la extensión *Blender Development*, la cual permitió desarrollar el trabajo sobre *Blender* con ayuda de herramientas de depuración y otras comodidades enfocadas en la creación de funcionalidades dentro del programa.

---

<sup>1</sup>Enlace al repositorio es:

<https://github.com/SergioJAlfonso/TFG-Distribution-System-of-Custom-Props-in-Blender>

## 3.1. Módulos del proyecto

Como se mencionó anteriormente en la Sección 1.3, uno de los objetivos principales es realizar una herramienta que pueda ser aplicada dentro de cualquier entorno de desarrollo tridimensional. Por ende, el funcionamiento de la extensión se encuentra dividido en dos módulos. El primero contiene los elementos directamente relacionados con el programa 3D, los cuales deben seguir cierta sintaxis, y se comunican con el mismo mediante su *API*. En el contexto actual, se referirá a este como el Módulo Blender. Por otro lado, el segundo módulo está relacionado con el funcionamiento principal de la extensión. Es decir, contiene los algoritmos y utilidades que realmente realizan la distribución, siendo totalmente ajeno al programa sobre el que se ejecute. Se referirá a éste como el Módulo Distribución.

Ambos módulos se comunican entre sí para llevar la ejecución tanto de la distribución como de las utilidades implementadas mediante el propio programa. En la siguiente sección se detallarán qué clases pertenecen a qué módulos y cómo estos se relacionan mutuamente, además de explicar el ciclo de ejecución de la herramienta.

## 3.2. Estructura de clases

Esta sección explicará brevemente las clases contenidas en cada módulo. Además, se desarrollará la forma en la que se relacionan entre ellas. Por último, se expondrá a grandes rasgos el flujo de ejecución de la herramienta.

### 3.2.1. Módulo Blender

Este primer módulo está compuesto principalmente por funcionalidades diseñadas expresamente para el programa 3D objetivo. El propósito principal es extraer y abstraer la información de los objetos existentes en dicho entorno, además de incorporar utilidades que facilitan el uso de la herramienta (véase Capítulo 5). Al usar *Blender* en este proyecto, la implementación de las funcionalidades están directamente ligadas con el uso de operadores que, según su definición en la Sección 5.1, se tratan de funciones que están registradas en *Blender* y se puede acceder a ellas mediante la interfaz gráfica. A continuación se detallarán las clases contenidas y sus correspondientes conexiones.

- **Operadores.** Éste módulo contiene múltiples clases que realizan distintas acciones, cada una con su respectivo operador. Dichas acciones están directamente relacionadas

con la distribución en sí y sus reglas (visitar Sección 4.6), y otras son meramente utilidades gráficas para facilitar el uso de la herramienta.

Las siguientes acciones son las más destacables para un mejor entendimiento:

- Ejecutar la función para distribuir uno o varios objetos sobre una superficie.
  - Eliminar los objetos distribuidos.
  - Asignar pesos a los vértices mediante un pintado de vértices (definido y detallado en profundidad en la Sección 5.2).
  - Definir distintos perfiles, cada uno con un conjunto de pesos de vértices distintos.
  - Ajustar los objetos a la normal del vértice sobre el que está posicionado.
  - Almacenar un conjunto de distintas distribuciones y alternar entre ellas (más detalles en la Sección 5.2).
- **Paneles.** Los paneles son el punto de entrada principal para ejecutar los operadores, pues estos deben estar sobre un panel para que el usuario pueda interactuar con ellos. Básicamente consisten de una zona en la interfaz gráfica donde es posible posicionar recursos de diseño de interfaces, como botones, cajas de texto, menús desplegables o deslizadores. En este caso, la extensión incorpora varios paneles para categorizar los diversos operadores, como se demuestra en la Sección 5.3.
- **BlenderUtils.** Se encarga de almacenar métodos para realizar tareas recurrentes relacionadas con elementos del propio programa, ya sean aplicadas a objetos tridimensionales o bien a componentes de la interfaz gráfica. Contiene funciones que se encargan de crear un cierto objeto en múltiples puntos del espacio 3D, de realizar una copia por referencia a un objeto y sus datos, y de extraer y analizar información del usuario a través de la interfaz gráfica. Asimismo, cuenta con otras acciones relacionadas con la interfaz gráfica de usuario, como la de inicializar una colección o incluso la de alternar entre distintas búsquedas de distribuciones. Estos últimos conceptos están definidos en la Sección 5.3.

### 3.2.2. Módulo Distribución

El segundo módulo, y más relevante de este trabajo, contiene el funcionamiento principal de la extensión. Es donde se encuentran los distintos algoritmos y estrategias que realiza en sí la distribución, utilizando estructuras externas como *AIMA* (ver Sección de 2.2.2), además de ciertas utilidades generales de uso frecuente para simplificar el desarrollo. A continuación se listarán los elementos contenidos en éste modulo y sus conexiones.

- **AIMA Problem.** Si bien es cierto que dicha clase no es propia de este proyecto, tiene un papel primordial en el funcionamiento de la herramienta. Se trata de una interfaz que representa un problema a solucionar. De esta manera, se puede implementar dicha interfaz para adaptarla a cualquier cuestión que se desee resolver. En este caso, la distribución de varios objetos sobre una superficie. En la Sección 4.3 se detallará en profundidad.
- **DistributionProblem.** Se trata de la clase que implementa la interfaz *AIMA Problem*. En ella se define la lógica y estrategia que se seguirá para solventar dicho problema.
- **Algorithm.** Para resolver un problema del tipo *AIMA Problem*, se debe hacer uso de algún algoritmo que sea compatible con los métodos definidos en el tipo mencionado. *AIMA* contiene una gran variedad de algoritmos que cumplen esta condición, sin embargo, para este proyecto, se han seleccionado y modificado los que mejor se adaptan al problema de distribución. En la Sección 4.3 se amplía este apartado.
- **StateDistribution.** En capítulos futuros se explica en detalle cómo funciona la distribución (véase Capítulo 4.4). No obstante, como idea general de a qué se refiere esta clase, se puede entender como un contenedor de información del estado actual del entorno. Almacena una lista de pares donde el primer elemento indica un vértice y el segundo indica si éste está ocupado, es decir, si hay un objeto posicionado sobre el vértice. Además, incluye en número de objetos distribuidos y el historial de acciones aplicadas al estado actual.
- **Action.** Determina el tipo de acción a aplicar. En el mismo capítulo en el que se expone cómo funciona una distribución se explica cómo es usada esta clase. En resumen, envuelve una serie de datos relacionados con el posicionamiento de un objeto, como su posición, rotación, escala y cuál es el objeto en cuestión (*Item*). Además indica si

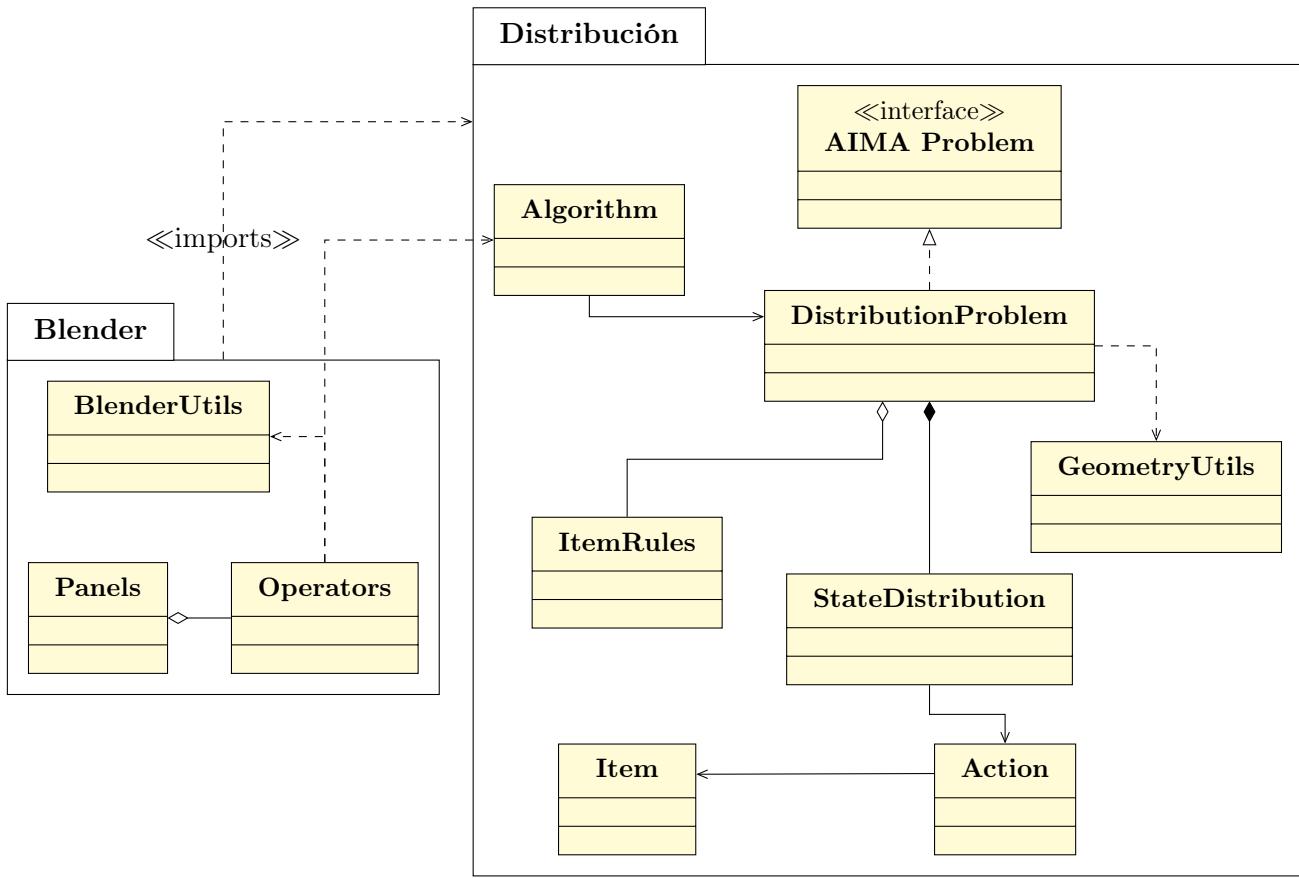
es una acción de creación o destrucción. Esta última tiene como objetivo revertir o modificar acciones de creación aplicadas anteriormente.

- **Item**. Como se explica anteriormente en la Sección 3.2.1, existen operadores para extraer y abstraer información del entorno 3D. La clase *Item* representa un objeto en dicho entorno, pero conteniendo datos cruciales que definen dicho objeto. Estos son su nombre, su posición, los datos de su malla y su volumen delimitador (véase Sección 4.6.3).
- **ItemRules**. De la misma forma que *Item*, *ItemRules* contiene toda aquella información proporcionada por el usuario, y está relacionado con qué reglas debe seguir la distribución sobre un objeto en particular. En el apartado 4.6.2 se detallan qué reglas se pueden definir.
- **GeometryUtils**. Por último, este fichero contiene código recurrente utilizado en la distribución de los objetos. En general, consiste de una serie de funciones dedicadas a extraer información en base a una serie datos de entrada. Algunos de estos pueden ser (1) la comprobación de que dos esferas y/o cajas se intersecan (relacionado con los volúmenes delimitadores), (2) filtrar una lista de vértices en función a un umbral, o (3) calcular el número de subdivisiones que permite una superficie en función de un volumen delimitador. Estas funciones son mencionadas y detalladas en la Sección 4.6.2.

### 3.3. Flujo de ejecución

Para concluir este capítulo es necesario explicar el flujo de ejecución de manera muy breve. Esto ayuda a comprender correctamente el motivo de las dependencias descritas en la sección anterior (3.2).

En la Figura 3.1 se muestra el diagrama *UML* que representa las clases descritas anteriormente, y donde se pueden apreciar las dependencias y relaciones entre ellas. Un diagrama *UML* es básicamente una manera para visualizar, especificar, construir y documentar sistemas de un programa y cómo se comportan [46]. Utiliza el Lenguaje Unificado de Modelado, que proporciona un conjunto de notaciones y técnicas para representar gráficamente los diferentes aspectos de un sistema [47].



**Figura 3.1:** Diagrama UML de la estructura del proyecto[48].

En primer lugar, al interactuar con un elemento contenido en un panel de la interfaz gráfica, se ejecuta la acción del operador asignado. Si se trata de una acción relacionada con *Blender*, potencialmente se hará uso de `BlenderUtils`, manteniéndose dentro del mismo módulo. En caso de que el operador accione la distribución en sí, se hará uso los elementos del Módulo Distribución. El algoritmo seleccionado resolverá un problema que, en este caso, consiste en posicionar los `Items` seleccionados de la manera indicada sobre una serie de vértices, siguiendo una serie de restricciones determinadas por `ItemRules`. El proceso será llevado a cabo mediante `StateDistribution` y las utilidades de `GeometryUtils`. Una vez concluido, la solución generada (una lista de objetos y sus posiciones) es devuelta Módulo Blender para poder ser interpretada y aplicada al entorno 3D.

# Capítulo 4

## Distribución de objetos sobre una superficie y definición de reglas

Este capítulo marca el comienzo del desarrollo del proyecto. Se tratarán las decisiones prematuras del proyecto y como éste fue evolucionando. También se abordarán en profundidad temas como: la creación de un primer prototipo de la distribución, la estructura de clases del proyecto y cómo se relacionan con un programa de gráficos tridimensionales, y la discusión sobre algoritmos de distribución y de búsqueda de caminos.

### 4.1. Implementación del primer prototipo

En las etapas iniciales del proyecto se propuso la creación de un prototipo inicial (véase Sección 1.3). El objetivo era lograr un resultado que cumpliera adecuadamente con los requisitos mínimos establecidos, acercándose lo suficiente a lo que se tenía planeado.

Para empezar, se precisaba de una idea inicial para acercarse a lo que esperado. Varias opciones fueron barajadas, tales como el algoritmo de *Poisson* [49], que se basaría para este caso en la división del espacio en celdas, generando así una matriz tridimensional que contuviese las posiciones de los objetos en función de aquel de menor tamaño. Sin embargo, esta opción no se ajustaba plenamente a la idea de posicionar objetos sobre una superficie. Se planteó pues, utilizar un elemento de la composición interna de la superficie como medio para determinar las posiciones de los objetos: los vértices. Se trata de uno de los elementos principales y más básicos de un objeto tridimensional virtual. Y del cuál se aprovecharán

muchas de sus características a la hora de distribuir.

Obtener los vértices de un objeto en programas gráficos implica más pasos que una mera llamada a la *API* (definida en la Sección 1.4). Para acceder a estos elementos y, por ende, a sus propiedades básicas, se necesita aplicar un arreglo matricial. En primer lugar, estos elementos forman parte de un objeto, lo que implica que sus coordenadas sean locales a dicho objeto. Esto significa que no se obtienen sus posiciones en el mundo, sino la posición relativa a la del objeto que pertenecen. Además, también hay que tener en cuenta que el objeto puede haber sufrido modificaciones en sus tres propiedades físicas. Por estos motivos es necesario aplicar una transformación para obtener sus coordenadas globales, lo que facilitará su manipulación y uso en futuras operaciones.

Una multiplicación matricial entre la matriz de transformación del mundo  $W$  y la matriz de posición  $P$  del vértice daría como resultado una matriz de posición en coordenadas locales  $P'$ . En la figura 4.1 se puede observar dicha operación. En el caso del programa *Blender*, la matriz  $W$  contiene las modificaciones físicas realizadas al objeto, tanto la rotación ( $R$ ), translación ( $T$ ), y escala ( $S$ ) en cada uno de los ejes. Estas modificaciones deber ser aplicadas para conocer la posición real del vértice en el mundo.

$$\begin{bmatrix} R \cdot Sx & R & R & Tx \\ R & R \cdot Sy & R & Ty \\ R & R & R \cdot Sz & Tz \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} \quad (4.1)$$

**Figura 4.1:** Transformación de coordenadas locales a globales.

Obtenido este resultado ya es posible aplicar una primera estrategia de distribución. La forma más rápida y sencilla es la de elegir al azar los vértices sobre los cuales se generarán objetos, consiguiendo así una primera solución del prototipo. El posicionamiento de los objetos, es una parte importante, pues no sólo se trata de crear instancias de un mismo objeto las veces que se necesiten. En muchos programas 3D, ésto implicaría la duplicidad de datos, ocupando más memoria y sobrecargando la escena de objetos idénticos. Al tener esto en cuenta, se implementó, con la ayuda de la *API* de *Blender*, una manera de crear instancias que sean copias por referencias. De esta manera, los objetos que aparentemente son copias de uno mismo, no son más que una referencia a una serie de datos esenciales (malla y textura), permitiendo así un resultado más eficiente y limpio.

Aún así, una distribución en base al método descrito es demasiado estricta, pues no da opción a personalizar la distribución. Idealmente, una distribución debe ser lo más flexible

posible a los cambios, pues en la mayoría de los casos no se trata de un problema con una única solución. La idea de usar vértices como puntos donde posicionar objetos da pie a utilizar la superficie como un lienzo. Con este concepto como base, se puede utilizar una técnica de selección de vértices a partir de asignación de pesos. A esto se le conoce como pintado por pesos, o comúnmente por su nombre en inglés *Weight Paint*. Ésta suele ser utilizada con distintos propósitos en el ámbito del modelado 3D. Entre los más empleados destacan la creación de esqueletos para personajes y su correspondiente animación, la aplicación de texturas sobre ciertas zonas de la malla o incluso la manipulación exclusiva de ciertos vértices. En el Capítulo 5, Sección 5.2 se detalla en más profundidad este concepto.

En el contexto de este proyecto, el pintado por pesos resultará de mucha utilidad dentro de la personalización de la distribución y en la ejecución de cálculos. Gracias a su rango  $[0, 1] \in \mathbb{R}$ , no sólo sirve para categorizar vértices, sino que también permite asignar influencia arbitraria a dichos vértices, siendo 0 la influencia nula y 1 la influencia máxima. Así, aquellos vértices cuya influencia supere cierto valor definido por el usuario serán considerados como candidatos en la distribución. Para mejor entendimiento, consultar la Sección 5.2)

Manteniendo el mismo esquema anterior de distribución, pero añadiendo el pintado por pesos, se consigue un nuevo enfoque más práctico y flexible. El número de objetos a distribuir está directamente ligado al número de vértices seleccionados que sobrepasen el umbral de influencia indicado. El pseudocódigo del Algoritmo 1 representa esta estrategia al completo.

Como resultado, el usuario es capaz de indicar sobre qué zonas de la malla potencialmente se distribuirán objetos. De esta manera se ha logrado un acercamiento funcional a lo que se estimaba en principio. Sin embargo, este enfoque es escueto, ya que no contempla posibles limitaciones que compliquen el proceso de distribución. Como, por ejemplo, evitar el solapamiento entre objetos o distancias mínimas entre los mismos. En el supuesto de que fuese contemplado, el diseño implementado puede, o no, encontrar una solución bajo los estándares pedidos. Si la encontrase, sería a base de varias ejecuciones, proceso el cual quita granularidad al resultado.

Conseguir un resultado que sea producto de una búsqueda exhaustiva implicaría diseñar un método para usar algoritmos de búsqueda de caminos, que utilice nuestras propias normas para guiar al algoritmo a un estado final considerado como meta. En la Sección 4.3 se profundizará sobre este tema.

---

**Algoritmo 1** Estrategia de distribución. Los vértices que superan un peso son elegidos para determinar al azar sobre cuáles de ellos se posicionarárán objetos.

---

**Require:**  $N \geq 0$  ▷ Número de objetos a distribuir  
**Require:**  $umbral \in [0, 1]$  ▷ Umbral límite de peso por vértice

*vertices*  $\leftarrow$  Conjunto de vértices y sus pesos  
*elegidos*  $\leftarrow$  Conjunto de vértices que sobrepasan el umbral  
**for each** *vertex* **in** *vertices* **do**  
    **if** *vertex.peso*  $>$  *umbral* **then**  
        *elegidos*  $\leftarrow$  **INSERT**(*vertex*, *false*, *elegidos*) ▷ *false* indica vértice no usado  
    **end if**  
**end for**  
*totalElegidos*  $\leftarrow$  Número total de vértices elegidos  
*N*  $\leftarrow$  **MIN**(*N*, *totalElegidos*)  
*solucion*  $\leftarrow$  Conjunto de vértices dónde posicionar objetos  
**while** *N*  $>$  0 **do**  
    *index*  $\leftarrow$  índice aleatorio de *elegidos*  
    **if** *elegidos[index]* no es usado **then**  
        *elegidos[index]*  $\leftarrow$  *true* ▷ *true* indica vértice usado  
        *solucion*  $\leftarrow$  **INSERT**(*vertex*, *solucion*)  
        *N*  $\leftarrow$  *N* - 1  
    **end if**  
**end while**  
**return** *solucion*

---

## 4.2. Ciclo de ejecución con *AIMA*

A lo largo de este documento se ha hecho mención a *AIMA* (definido en la Sección 2.2.2) y su papel crucial en el desarrollo de este proyecto. Esta sección pretende explicar el funcionamiento interno de *AIMA* y cómo fue aprovechado para la distribución de objetos sobre superficies. Sin embargo, es importante recalcar que no se abarcará todo el contenido del libro, pues su extensión va más allá de lo que se ha usado en la herramienta. Por ende, se explicarán únicamente los principales componentes empleados y el motivo de su uso.

### Problem

Para llevar a cabo la resolución de una tarea, *AIMA* la encapsula en “problemas”. En la Sección 3.2.2 también se hace mención de la clase `Problem`. Esta es una clase interfaz que debe ser implementada por el programador con el objetivo de adaptarla al problema en cuestión [1].

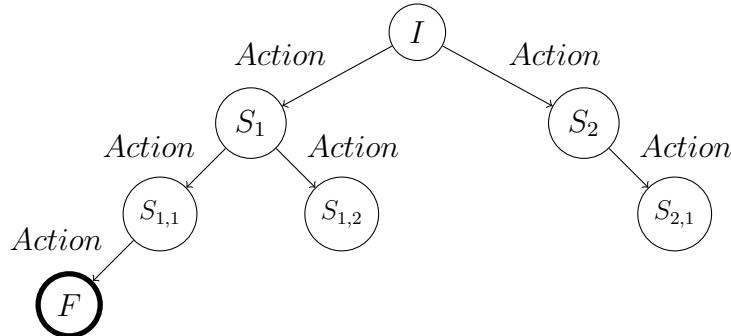
La interfaz `Problem` se encarga en esencia de procesar dos datos de entrada. Ambos representan el estado del problema, pero con la diferencia de que uno de ellos es el estado inicial, y el otro el estado al que se quiere llegar, conocido como estado final. Además, contiene una serie de métodos públicos que inicialmente están vacíos o almacenan código temporal de ejemplo, también expuestos a las decisiones del programador. Al implementar esta interfaz se deben redefinir varios de estos métodos para establecer el comportamiento deseado del problema, lo que le permitirá alcanzar el estado final. Formalmente, los métodos principales son:

- **Acciones.** Dado un estado particular, el método devolverá una lista de posibles acciones a tomar en base a dicho estado y que permitan alcanzar un estado sucesor. En caso de que no haya acciones posibles, el estado es descartado. Posteriormente, en la Sección 4.4 se explicará cómo esta función toma un papel principal en la búsqueda de soluciones.
- **Resultado.** En relación con el método anterior, este paso se encarga sencillamente de aplicar una acción a un estado en concreto. Esto producirá un nuevo estado, el cuál será devuelto para continuar el ciclo.
- **Prueba Objetivo.** Dado un estado, éste es comparado con el estado final. Si ambos coinciden, la búsqueda de soluciones concluirá; de lo contrario, se continuará la bús-

queda. En caso de concluir, se retornará la lista de acciones que han sido aplicadas al estado inicial para llegar hasta el estado final, es decir, un camino. Esta función es llamada después de la función Resultado.

En la resolución de un problema, estos tres métodos deberán ser, teóricamente, ejecutados en orden de manera indefinida hasta que se alcance un estado final. Sin embargo, esto puede desembocar en una búsqueda infinita, en la que desde el estado inicial nunca se encuentre la manera de llegar a un estado final. Esto desemboca en la necesidad de implementar medidas de seguridad, que acompañadas de un buen diseño del problema, permitan evitar resultados indeseados o posibles errores durante la resolución del problema. Y, que en caso de producirse, se mantenga en todo momento el control de la situación.

Analizando este ciclo de ejecución se puede llegar a la conclusión de que progresivamente se está formando un grafo o un árbol de nodos. Cada estado representa un nodo, y cada acción una arista que apunta a otro nodo, tal y como se ve en la Figura 4.2. De esta manera, algoritmos de búsqueda de caminos, tales como búsqueda en anchura o en profundidad, pueden ser aplicados.



**Figura 4.2:** Árbol de nodos. El nodo raíz  $I$  es el estado inicial. Los nodos  $S$  son nodos sucesores producto de una acción (Action). El nodo  $F$  es el estado final, considerado como solución.

AIMA, además de proporcionar `Problem`, también recopila una gran variedad de algoritmos enfocados en múltiples campos. Uno de ellos es el recorrido de estructura de datos complejas, como los árboles binarios o grafos dirigidos/no dirigidos [50].

Anteriormente se mencionaron tres principales métodos que deben ser definidos. Sin embargo, existen otros dos los cuales, según el algoritmo, serán utilizados para guiar el estado a una solución más prometedora, o por el camino más corto. Estos dos son:

- **Coste del Camino.** Dados dos estados, siendo  $B$  el sucesor de  $A$ , se calcula el costo del camino desde  $A$  hasta  $B$ . Este nuevo coste se asigna a  $B$ . Este proceso se repite con el resto de nodos a medida que van surgiendo.
- **Valor del Nodo.** En función del problema, esta función devolverá un valor que representa el estado actual. Dicho valor será potencialmente maximizado por algoritmos como *Simulated Annealing* (véase Sección 2.2.2).
- **Heurística.** Algunos algoritmos de búsqueda informada, tales como A\* (leído como A-Estrella), utilizan esta función para encaminar el estado actual a la solución más prometedora, obviando así caminos más largo o aquellos que posiblemente no lleven a una solución.

De esta manera, para resolver un problema se puede hacer uso de algoritmos recopilados en *AIMA* y explicados en la Sección 4.3. Más adelante en la Sección 4.4 se hablará de cómo fueron adaptados a la herramienta. Además, se explicará cómo se redefinieron los métodos descritos para conseguir una distribución de objetos sobre una superficie.

## Node

A parte de `Problem` y numerosos algoritmos de inteligencia artificial, también se encuentra la clase `Node`. El propósito principal de esta clase es la comunicación entre el problema actual y el algoritmo utilizado, el cual será el encargado de producir el ciclo de ejecución para la resolución del problema.

La solución final, como se mencionó en el método Prueba Objetivo, se conforma de la lista de acciones consecutivas que se fueron aplicando hasta llegar al estado final. Esto es gracias a la clase `Node`, la cual incorpora distintas funcionalidades para obtener información acerca del nodo en sí. Principalmente almacena un estado, su nodo predecesor (nodo padre), y la acción que lo generó. Algunas de sus funcionalidades más destacables son:

- Generar un nuevo nodo hijo dada una acción.
- Devolver una lista de nodos alcanzables desde el nodo.
- Devolver una lista de nodos predecesores al nodo.

Para resumir este capítulo, habiendo diseñado el problema a solventar, y habiendo escogido un algoritmo en base a su funcionamiento y adaptabilidad, el ciclo de ejecución consta

---

**Algoritmo 2** Algoritmo de Búsqueda en Profundidad que realiza el ciclo de ejecución para solventar un problema dado. La función *Solution* devuelve una lista de acciones del nodo final.

---

```

Require: Problem                                ▷ Problema a solucionar
Require: nodoIni                            ▷ Nodo con estado inicial
          porVisitar                         ▷ Pila de nodos que por explorar
          porVisitar  $\leftarrow \text{INSERT}(\text{nodoIni}, \text{porVisitar})$ 
while porVisitar no está vacío do
    nodo  $\leftarrow \text{POP}(\text{porVisitar})$ 
    if PRUEBA_OBJETIVO(nodo) then
        return SOLUTION(nodo)
    end if
    nodosHijos  $\leftarrow \text{EXPAND}(\text{nodo}, \text{problem})$            ▷ Genera los nodos sucesores
    porVisitar  $\leftarrow \text{INSERT}(\text{nodosHijos}, \text{porVisitar})$ 
end while

```

---

de (1) obtener una lista de acciones posibles desde el estado inicial; (2) generar tantos nodos hijos como acciones posibles; (3) evaluar cada nodo, y si coincide con el estado final, devolver la solución; o (4) de lo contrario, volver a ejecutar el mismo proceso desde el paso 1 por cada nodo hasta encontrar una solución. En el Algoritmo 2 se puede ver un ejemplo de este proceso usando un algoritmo de Búsqueda en Profundidad.

### 4.3. Algoritmos de búsqueda de caminos para distribuir objetos

El problema que pretende resolver este trabajo de investigación es el de distribuir elementos tridimensionales sobre una superficie. Una solución básica consistiría simplemente en elegir posiciones del escenario al azar sobre los que llevar a cabo la distribución. No obstante, esta solución está lejos de ser ideal desde un punto de vista práctico. Por ello, este trabajo está diseñado para permitir al usuario especificar condiciones y reglas que aumentan la complejidad de este proceso. Por ejemplo, se busca dar la posibilidad de que los objetos puedan (o no) colisionar; o que exista al menos  $N$  unidades de distancia mínima entre objetos. Sin embargo, aún hace falta una manera de permitir distintas formas de resolver una distribución.

Esta carencia es el motivo para dar la opción de elegir un algoritmo. Tal como se señaló en la Sección anterior 4.2, para solventar este problema usando *AIMA* se puede hacer uso

de algoritmos de búsqueda de caminos a la hora de buscar una solución. No obstante, cada algoritmo tiene un comportamiento específico según para qué se use. Se podría usar una búsqueda por vuelta atrás, búsqueda en profundidad, o incluso un algoritmo más sofisticado como A\* (véase Sección 2.2.2).

Distribuir objetos en sí es elegir una posición, comprobar que el objeto posicionado sigue las normas especificadas, y repetir este proceso hasta que se llegue al número de objetos deseados. Por lo que este procedimiento se puede resumir en: dada una superficie sin objetos posicionados (estado inicial), ir generando variantes de esta superficie (estados sucesores) a base de posicionar objetos (posibles acciones), y por cada variante, comprobar si ésta contiene el número de objetos deseados (estado final). Teniendo este concepto en mente, surge otra pregunta: dados dos estados cuyo número de objetos posicionados sea el mismo, pero que hayan sido distribuidos de manera desigual, ¿cómo se sabe cuál de los dos está mejor encaminado a la solución final? En términos generales, ambos tienen la misma probabilidad de llegar al estado final, por lo que no es una tarea sencilla estimar, mediante una heurística, cuánto le queda al estado actual para llegar al final. Esto se debe a que dicha estimación sería similar para todas aquellas superficies con  $M$  objetos, siendo  $M$  menor al número de elementos a distribuir. Por ende, en principio el uso de A\* queda descartado, ya que es un algoritmo informado, mientras que nuestro problema otorga potencialmente la misma información a distintos estados.

En un principio se planteó, e incluso llevó a la práctica, el uso del algoritmo de búsqueda en anchura (comúnmente conocido como *BFS* por sus siglas en inglés). El cuál explora el espacio de búsqueda priorizando aquellos nodos (estados) menos profundos (en este caso, con menos objetos). El uso de este algoritmo puede parecer que llegará, eventualmente, a la solución. El principal inconveniente es el tiempo que puede llegar a tardar. Los nodos menos profundos del árbol (que están más cerca de la raíz), se expandirán antes que los nodos más profundos (más lejanos a la raíz). Esto implica que el algoritmo pueda llegar a tardar un tiempo considerable en alcanzar la solución, aumentando directamente con el número de objetos a posicionar.

Hacía falta un algoritmo que se centrara en avanzar continuamente hasta llegar a la solución final, priorizando aquellos estados que tengan más objetos correctamente posicionados. El algoritmo *Escalada Simple* (conocido en inglés como *Hill Climbing*) se comporta de esta manera (véase Sección 2.2.2), priorizando siempre aquel nodo cuyo coste sea mayor al actual. Refiriéndose por coste al valor que tiene asignado el estado actual (véase Sección

anterior 4.2). Además, el tiempo empleado en encontrar el estado final es relativamente pequeño, pues el espacio de búsqueda se limita a aquellos nodos con valor superior al actual. Los dos únicos inconvenientes de este algoritmo son:

- Si no existe ningún nodo sucesor con valor superior, el nodo actual será considerado como solución final. Esto no es del todo cierto, pues no ha alcanzado el número objetivo de elementos especificados. Es decir, ha llegado a una solución *local*, no global.
- Al intentar maximizar el valor de *Node*, no se da opción a considerar otros nodos que podrían llegar a la solución global. Por lo tanto, no hay forma de encontrar varias soluciones en un solo recorrido.

A pesar de estos inconvenientes, este algoritmo se utilizaba para la resolución del problema principalmente por su eficiencia y velocidad a la hora de encontrar una solución. Aun así, una solución parcial/local no garantizaba seguir las especificaciones mínimas del número de elementos a colocar. Esto llevó a buscar un segundo algoritmo similar para compensar esta desventaja. Como se mencionó en el Capítulo 2.2, el *Temple Simulado* (conocido en inglés como *Simulated Annealing*) pretende cubrir este inconveniente. Al encontrar una solución local, aleatoriamente y en base a una probabilidad, elige un nodo sucesor, incluso si su valor es menor. De esta manera tiene una nueva oportunidad de encontrar la solución global.

Estos dos algoritmos dieron buenos resultados a la hora de solucionar el problema de distribución de objetos (más detalles del problema en la siguiente Sección 4.4). Proporcionaban resultados rápidos y la mayoría de veces posicionaban la cantidad exigida. Aunque seguía habiendo situaciones en las que encontraban soluciones parciales. Por ello, hacía falta un tercer algoritmo. Uno que fuese capaz de explorar en más profundidad el espacio de búsqueda, pero que a la vez priorizara ciertos nodos para llegar a la solución.

Si bien es cierto que A\* fue descartado anteriormente, no toda su estrategia tenía que serlo. A\* no es más que un caso particular del algoritmo *Búsqueda del Primero Mejor* (conocido por sus siglas en inglés *BFS*). A diferencia de A\*, que utiliza el coste del camino y la función heurística del propio problema, *BFS* sólo utiliza una heurística para llegar a la solución. Sin embargo, en principio no es posible proporcionar una función que estime el coste del camino más pequeño hasta el estado final. Lo que sí se puede emplear es una variante de éste conocida como *Búsqueda de Coste Uniforme*. Se trata del mismo algoritmo, salvo que la función heurística consiste básicamente en minimizar el coste del camino desde el nodo inicial hasta el nodo actual. Esto quiere decir que, redefiniendo el método Coste del

Camino del problema, se puede diseñar un factor que analice el estado (nodo) y penalice a aquellos estados con pocos objetos, con un coste muy elevado. Mientras que aquellos con más objetos posicionados son recompensados con un coste reducido.

De esta manera, se obtiene un algoritmo que, aparte de explorar el espacio de soluciones en gran medida, y por ende, conseguir llegar a una solución global, también abre la posibilidad a encontrar más de una solución en la misma búsqueda. Dos de los inconvenientes que presentaban los dos algoritmos previamente descritos.

En definitiva, se han seleccionado 3 algoritmos que a simple vista se adaptan al problema de este trabajo de investigación: (1) *Escalada Simple*, (2) *Temple Simulado* y (3) *Búsqueda de Coste Uniforme* (variante de *Best First Search*), y que además han demostrado obtener resultados satisfactorios. Es importante destacar que este último fue alterado para devolver más de una solución al problema (visitar Sección 4.3.1).

#### 4.3.1. Adaptación del algoritmo para múltiples soluciones

Como se mencionó al final de la Sección Algoritmos de búsqueda de caminos para distribuir objetos, *Búsqueda de Coste Uniforme* (variante de *BFS*), fue alterado con el objetivo de que pueda devolver más de una solución. Se recuerda que una solución se representa como una lista de acciones aplicadas hasta llegar al estado final (véase Sección 4.2).

Para empezar, el algoritmo no sólo recibe un *problema* a resolver como entrada inicial, sino que también recibe un número entero natural que representa el número de soluciones que debe encontrar en su búsqueda. En consecuencia, la búsqueda tiene dos condiciones de parada: mientras existan nodos por visitar, y aún no se hayan encontrado las soluciones pedidas. Así, cada vez que se encuentra una solución, ésta es almacenada en una lista que al finalizar el algoritmo será devuelta como conjunto de soluciones.

Esta modificación es sencilla de implementar, pues sólo implica modificar ligeramente la estructura del código original. Esto es, añadir la nueva condición de parada y adaptar el algoritmo para que recoja una lista que almacene las distintas soluciones. Sin embargo, como se comentó previamente en la Sección 4.3, no es posible aplicar dicha alteración a los otros dos algoritmos usados (*Temple Simulado* y *Escalada Simple*), ya que éstos sólo se centran en maximizar un solo nodo, sin tener en cuenta otros posibles caminos. Cambiar este comportamiento sería variar radicalmente la naturaleza del algoritmo y perdería gran parte de sus cualidades.

Por último, al final de la Sección 5.3 se comenta cómo esta funcionalidad se relaciona con el programa 3D. Además, en la Sección 4.4 se habla de cómo tuvo que ser diseñado el Problema para permitir que siempre exista al menos el número de soluciones pedidas.

## 4.4. Diseño del problema

Esta sección pretende explicar cómo se diseñó e implementó la clase principal de este trabajo, conocida como `DistributionProblem`. Se trata de una clase que implementa la interfaz `Problem`. En la Sección 4.2 se señala que cualquier problema que se desee solventar usando algoritmos recopilados por *AIMA*, debe implementar esta interfaz y sus métodos principales, con tal de hacer funcionar el ciclo principal de ejecución.

Por otro lado, anteriormente en la Sección 4.3 se nombran los algoritmos que se usarán para la distribución de objetos sobre una superficie. Por ello, la redefinición de algunos métodos se debe al uso de ciertos algoritmos que requieren de éstos. Sin embargo, se detallarán más adelante, pues se empezará explicando los 3 componentes básicos que se redefinieron, y cómo está representado el estado del problema (véase Sección 4.2, donde se define en profundidad los métodos a redefinir).

En primer lugar, la gran mayoría de métodos a implementar están directamente ligados con un estado como parámetro de entrada. El estado, en el contexto de este problema, se representa como una lista de pares, donde el primer elemento indica un vértice y el segundo indica si éste está ocupado. Adicionalmente contiene también cuántos vértices están ocupados, así como un historial de acciones aplicadas a este estado en concreto. Este estado está abstracto en la clase `StateDistribution` (véase Sección 3). Así, dado un estado, el método que se encarga de indicar si éste coincide con el estado final (estado objetivo) es el método *Prueba Objetivo*. El estado inicial y final difieren ligeramente. Mientras que el primero se trata de un estado vacío, sin vértices ocupados ni acciones aplicadas (en principio), el estado final consta la misma lista vacía de vértices, a excepción de que el contador de números de objetos coincide con el número de objetos a posicionar. Esta decisión se tomó debido a que, como desarrolladores, no hay manera de saber cuál es el estado final que desea el usuario. Pero si se puede saber cuántos objetos quiere que existan en el entorno. De esta manera, en la función *Prueba Objetivo*, basta con comparar este atributo de los dos estados. Si coinciden, se ha alcanzado el objetivo; de lo contrario, se ha de continuar iterando.

En segundo lugar, el método *Resultado*, se encarga de aplicar una acción a un estado

y generar así uno nuevo. En la Sección 3.2.2 se explica en qué consiste una acción en este problema. Básicamente consta de un índice que indica qué vértice está ocupado, además de qué transformaciones físicas sufre el objeto posicionado, tales como rotaciones o escalado en varios ejes. Por lo tanto, al aplicar una acción se trata de indicar que el vértice  $i$  está ocupado, y aumentar así el contador de objetos posicionados. No hay que almacenar las transformaciones físicas, pues el nodo en sí contendrá qué acción lo generó y, por consiguiente, esta información. Es importante destacar que, en la sección mencionada, se habla de acciones que son capaces de deshacer otras acciones. Éstas principalmente contienen qué vértice debe ser “liberado”, y una etiqueta que indica que son una acción para revertir.

Y por último, siendo el método más importante, *Acciones*. Dado un estado, éste debe ser analizado para así determinar qué acciones se pueden aplicar sobre el mismo. Esto se realiza para posteriormente devolver una lista de acciones. Para este problema se ha diseñado un método que realice las siguientes tareas:

- Garantizar que al menos habrá el número de soluciones demandadas por el usuario (véase Sección 4.3.1).
- Retornar acciones, si y solo si, éstas pueden ser aplicadas siguiendo las restricciones del usuario (véase Sección 4.6).
- Que no existan acciones repetidas.

Empezando desde la última, esta tarea es llevada a cabo implícita y explícitamente. Implícitamente ya que no se puede crear una acción que ordene posicionar un objeto en un vértice ya ocupado, pues sería absurdo y el estado en sí no sufriría ningún cambio; no habría nodo hijo. Y, por otro lado, es explícito ya que al determinar qué acciones pueden ser aplicadas, éstas son almacenadas en una lista. De esta manera, al crear una acción que en principio ha comprobado que el índice  $i$  del vértice está libre, adicionalmente se comprueba que ninguna acción de la lista previamente mencionada no contiene un índice  $j$  tal que  $i = j$ . Formalmente, siendo  $V$  un conjunto de índices de vértices de acciones y  $A$  una acción recién generada. Se debe cumplir que:

$$\{\forall j \in V; i \in A \mid i \neq j\}$$

Por otro lado, una acción generada debe ser comprobada para evaluar si las transformaciones físicas producidas y la posición del objeto cumplen con las normas establecidas por el usuario. Básicamente, la acción es analizada por una función que se encarga de devolver una

variable booleana con el resultado. En caso de que se cumpla, la acción es almacenada en la lista y se procede a seguir generando acciones. De lo contrario, se descarta. En la Sección 4.6.4 se explica más a fondo cómo funciona esta comprobación.

En tercer lugar, la generación de acciones es un factor crucial para la distribución de objetos, pues sin ellas no se podría alcanzar la solución final. Anteriormente, en la Sección 4.2, se mencionó que el proceso de buscar una solución a un problema genera en sí un árbol de nodos. Este árbol de nodos puede crecer más o menos en función del número de hijos que obtenga cada nodo. Suponiendo que se desean distribuir 15 objetos, y que cada estado puede generar 3 acciones, cada una posicionando un objeto, se generará un árbol de altura  $h = 15$ , con 3 hijos por cada nodo. La formula de la Figura 4.3 permite saber el número de nodos, conociendo la altura y el número de hijos por nodo. Así, el número total de nodos (el espacio de búsqueda) es de 21523360 nodos. Este número es considerablemente grande para una sencilla tarea como es posicionar objetos. Además, no se ha tenido en cuenta el número de vértices disponibles ni tampoco la comprobación de restricciones por cada acción. Por lo que, se puede intuir que el número de acciones a generar provocará un gran impacto en la búsqueda del estado final. Asimismo, en función del número de soluciones que especifique el usuario, el último nivel de este árbol debe ser mayor o igual a dicho valor, pues en el último nivel se han posicionado los objetos pedidos. En el ejemplo anterior, el nivel 15 del árbol tiene  $3^{15}$  nodos, es decir, 14348907 nodos.

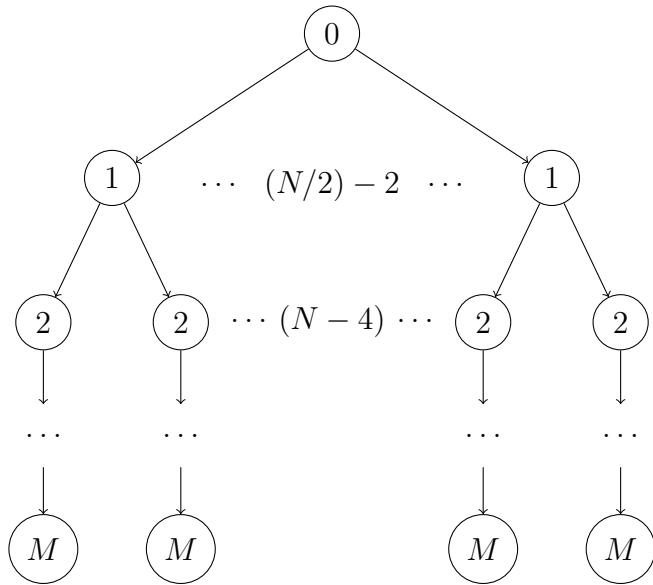
$$\frac{B^{h+1} - 1}{B - 1}$$

**Figura 4.3:** Fórmula para calcular el número de nodos de un árbol, conociendo su altura  $h$ , y el número de hijos por nodo  $B$ . [50]

Otro inconveniente se suma a los anteriores. Siendo un árbol de altura 2 y con 3 hijos por nodo, se tiene un total de 13 nodos. En el último nivel existen 9 posibles soluciones. Supongamos que el usuario ha pedido 9 soluciones. Estos 9 nodos provienen de 3 nodos padre, por lo que son una variante de éste. Esencialmente varían en 1 objeto, ya que cada acción posiciona 1 objeto. Por consiguiente, las 9 soluciones obtenidas se ven reducidas realmente a 3 soluciones, en las que hay 3 variantes de cada una, modificando sólo un objeto. Lo que quiere decir que en realidad no hay soluciones variadas.

En un principio no se puede saber cuántos hijos hay que generar por nodo para encontrar

la solución, y mucho menos los suficientes para que exista más de una solución. Por otro lado, también hay que garantizar que cada solución es distinta a cualquier otra. Si se analiza bien, en la primera generación de nodos (hijos del nodo raíz), éstos están muy diferenciados entre sí, ya que en un principio no hay objetos posicionados. Posteriormente se van generando variantes de esta generación, en las que se van fijando ciertas posiciones hasta conseguir generaciones que se parecen mucho entre sí. En definitiva, la solución que se tomó radica principalmente en garantizar que por lo menos hay  $N$  soluciones, siendo  $N$  el número de distribuciones demandadas por el usuario. Para ello, el nodo raíz (estado inicial) generará  $\frac{N}{2}$  acciones distintas, resultando en  $\frac{N}{2}$  nodos en la primera generación. Cada uno de éstos generará 2 nodos, obteniendo así un total de  $N$  nodos en tan solo el nivel 2 del árbol. Por otra parte, estos nodos no pueden seguir produciendo dos hijos, pues esto llevaría al inconveniente anterior de soluciones poco variadas entre ellas. Simplemente ahora cada nodo generará un solo hijo, hasta llegar al estado final. La Figura 4.4 representa un esquema de esta estrategia.



**Figura 4.4:** Representación gráfica del árbol de nodos generado al crear acciones que permitan garantizar el número de soluciones solicitadas  $N$ , así como variadas. El número de cada nodo indica cuántos objetos han sido posicionados, siendo  $M$  el número del estado final.

Con esta estrategia se ha solventado, teóricamente, obtener el número de soluciones demandadas por el usuario e incluso que éstas sean variadas. Además, a partir del nivel 2, el proceso es más rápido, pues no hay bifurcaciones en los nodos posteriores. En el caso de que un estado no tenga hijos, fruto de no poder cumplir las restricciones, se haría uso de

la acción de tipo reversión (véase Sección 3), la cual simplemente revertirá aleatoriamente cualquier acción aplicada previamente.

Para finalizar, la creación de una acción es relativamente sencilla. Simplemente consta de elegir al azar un vértice del conjunto de los vértices libres. Además, en función de las reglas del usuario (permitir rotaciones o escalados en ciertos ejes bajo ciertos rangos), se procederá aplicar, aleatoriamente una rotación y/o escalado en los ejes especificados, siguiendo unas cotas mínimas y máximas. Como se mencionó anteriormente, esta acción es puesta bajo evaluación para comprobar que cumple las restricciones pedidas (ver Sección 4.6.3). En caso de que no los cumpla, se descarta y se genera una nueva acción. Para más información sobre las reglas y restricciones visitar la Sección 4.6.

## 4.5. Redefinición de los métodos Valor y Coste del Camino

Algunos métodos deben ser redefinidos para que otros algoritmos hagan uso de ellos. En el contexto actual, dos de los tres algoritmos elegidos (véase Sección 4.3) utilizan la función Valor para su funcionamiento. Estos son *Temple Simulado* y *Escalada Simple*. La función les sirve de guía para saber qué camino tomar para llegar a la solución. Como el objetivo de estos algoritmos es maximizar dicho valor, se decidió por redefinir esta función simplemente haciendo que retorne el número de objetos posicionados hasta ahora.

Por otro lado, el algoritmo *Búsqueda del Primero Mejor* utiliza el coste del camino del estado con el mismo objetivo: llegar a la solución. Por ende, se decidió que, conforme más objetos tuviera el estado, menos coste obtendría. Por el contrario, a menor cantidad de objetos, más coste. Dicho coste se adecua a la cota  $c \in [0, 100]$ . Sin embargo, hay que tener en cuenta que, si 10 estados tienen el mismo número de objetos, coincidirá también el coste. En relación a la manera de distribuir objetos, se pueden aplicar las mismas reglas de elección al azar para solucionar este inconveniente, pero dentro de unos límites. En general, se realiza una relación entre la cantidad de objetos posicionados y los que se van a colocar. Esto produce un número contenido en el rango  $[0, 1] \in \mathbb{R}$ , que se entiende como porcentaje de objetos posicionados. A continuación, se obtiene el porcentaje de objetos por colocar (referido como X), restando 1 al porcentaje anterior. Por último, se genera un número aleatorio entre  $[0, X * 100]$  que se considerará como el costo a asignar. De esta manera, conforme más objetos falten por colocar, mayores serán las probabilidades de

que el coste sea mayor. Al contrario, entre más objetos estén colocados, menor será el coste asignado. Estas estrategias han mostrado funcionar satisfactoriamente tanto con *Hill Climbing* y *Simulated Annealing*, como con *BFS*.

## 4.6. Definición de reglas y sus efectos en la distribución

En esta sección se van a desarrollar las reglas y propiedades que se pueden definir en cada objeto para modificar su comportamiento en la distribución. Se hablará sobre cuáles se han decidido y por qué, así como de la manera en la que se pueden modificar para obtener diferentes resultados.

Posteriormente se especificará de qué manera afectan a la generación del escenario y cómo interactúan entre ellas. Por último, se profundizará en todas las limitaciones que presentan.

### 4.6.1. Planteamiento original del sistema de reglas

Uno de los objetivos con mayor importancia a la hora de completar el diseño de este trabajo era definir claramente una serie de reglas y restricciones que permitieran delimitar la forma en la que los objetos son colocados, tanto individualmente como en relación al resto de objetos de la distribución. Estas reglas se comenzaron a desarrollar una vez se contaba con los primeros prototipos de distribución. Pudiendo apoyarse en estas primeras iteraciones del proyecto, se abordó la manera en la que se pondrían en práctica.

Originalmente, debido a la variedad de objetos que podrían distribuirse, se enfocó el diseño de las reglas como perfiles de objetos que compartirían restricciones específicas. Un ejemplo de esto sería un perfil para elementos de un entorno natural, como árboles y piedras, o uno diferente para mobiliario de una casa, como sillas y mesas. Estos conjuntos de objetos se distribuirían teniendo en cuenta restricciones parecidas. De esta forma, sería sencillo aplicar un perfil a un elemento, de forma que ese elemento tuviera que seguir las restricciones aportadas por ese perfil. Si bien se trata de un diseño que simplificaría el flujo de trabajo del usuario, supone también una gran limitación. Este formato de reglas obliga a que los objetos sean encapsulados en un grupo de restricciones finito, que no da pie a distribuir objetos que se salgan de los perfiles contemplados.

Teniendo en cuenta que el diseño anterior resulta muy restrictivo, se dio paso al final-

mente implementado. La conclusión que se alcanzó era que cada objeto tendría el mismo conjunto de reglas, y el usuario debería poder modificar las reglas de cada objeto por separado. Teniendo en cuenta que este sistema tiene escalabilidad, es decir, que se pueden definir tantas reglas como se considere necesario, se plantearon las primeras reglas.

#### 4.6.2. Reglas sobre las propiedades del objeto

Existen tres propiedades principales que conforman la situación de un objeto dentro de un espacio tridimensional. En la sección previa, se ha hablado de la *posición* de los objetos en la superficie. La posición es la propiedad más importante que se debe fijar cuando se distribuyen los objetos en un entorno. Los algoritmos de distribución mencionados en el capítulo anterior se encargan de establecer esta propiedad en cada uno de los objetos distribuidos para generar un entorno diverso.

Sin embargo, para generar un escenario variado y convincente, se debe tener en cuenta que los elementos distribuidos deberían tener una orientación diferente entre ellos, especialmente si son del mismo tipo. Asimismo, cada elemento debería tener un tamaño distinto si se pretende alcanzar una diversidad en el escenario. Por ello, cada objeto tendrá que tener en cuenta dos factores importantes, la *rotación* y la *escala*.

Para alcanzar un resultado que el usuario pueda configurar a su preferencia, estas propiedades se han desglosado en varias reglas. Dado que estas reglas afectarán principalmente a la forma en la que se presenta un objeto en el escenario, nos referiremos a ellas como reglas propias de cada objeto. Estas reglas disponen tanto de un parámetro por defecto, como de uno o varios límites que el parámetro asociado a ellas no puede superar. Una vez definidas correctamente, el algoritmo de distribución tendrá en cuenta las transformaciones que debe aplicar a cada objeto. Las características más concretas de estas reglas se detallan en la Tabla 4.1.

En cuanto a la rotación, se han establecido tres reglas. La primera permite al usuario seleccionar en cual de los tres ejes cartesianos se permite la rotación del elemento. Las siguientes dos son accesibles una vez se decide si se puede aplicar una rotación al objeto. Éstas son (1) el rango máximo de rotación que permite el objeto, y (2) la cantidad en grados en la que se aumentará o disminuirá la rotación.

Con relación a la escala, se cuenta con 2 reglas: El mínimo factor de escala aplicable al objeto y el máximo. Cada vez que se genere una instancia del objeto en el escenario, se

aplicará un factor aleatorio entre esos 2 de manera uniforme en todos los ejes. Al contrario que al realizar una rotación, no interesa diferenciar los ejes por separado, pues eso implicaría deformar la malla del elemento distribuido. A pesar de que el usuario pudiera encontrar útil esta diferenciación, se ha considerado que estas deformaciones serían causa de una complejidad añadida que se aleja del objetivo del trabajo.

Hay que destacar una última regla propia de cada objeto, el *peso de aparición* (no hay que confundirlo con pintado por pesos o *Weight Paint* mencionado en la Sección 4.1). Esta regla define un peso que establecerá si, en un grupo con diferentes tipos de objetos, se generarán más o menos instancias del objeto cuya regla se modifique. Al realizarse la distribución, se tendrán en cuenta los pesos de cada objeto a distribuir, habiendo así una mayor o menor probabilidad de que aparezca cada uno de ellos.

Por último, existe una regla que es común a todos los objetos, sin diferenciar entre ellos, el *ajuste a la normal*. Se ha decidido hablar de ella en este apartado debido a que se trata de una regla que afecta a cada elemento por separado, es decir, no tiene en cuenta la posición ni estado del resto de objetos que se distribuirán. En geometría, una normal es una recta que es perpendicular a otra dada en un punto de intersección o tangencia. En el contexto del modelado 3D, las normales se utilizan para determinar la iluminación y los reflejos en un escenario. En general, una normal que apunta hacia afuera desde la superficie es lo que se espera para la mayoría de las aplicaciones de modelado, ya que la luz se reflejará de manera más realista. En este caso particular, la normal permite establecer si un objeto se debe ajustar, en menor o mayor medida, a la inclinación de la superficie sobre la que se encuentra. Un ejemplo de elementos que no deberían ajustarse a la normal en gran medida son objetos como postes de luz o vallas, pues no se colocarán de manera perpendicular al terreno si ello afecta a su función.

Regla	Tipo	Por defecto	Límites	Descripción
Allow Rotation	Lista de enteros	[0,0,0]	$\{\{0,1\}, \{0,1\}, \{0,1\}\}$	Define si se aplicará rotación al objeto o no en los ejes $X$ , $Y$ y $Z$ respectivamente.
Rotation Range	Lista de puntos flotantes	[180.0, 180.0, 180.0]	$\{\{0,180.0\}, \{0,180.0\}, \{0,180.0\}\}$	Define el máximo ángulo de rotación del objeto en grados en cada uno de los ejes entre $-j$ y $j$ , siendo $j$ el parámetro de cada eje ( $X$ , $Y$ y $Z$ ).
Rotation Steps	Lista de puntos flotantes	[1.0, 1.0, 1.0]	$\{\{0,180.0\}, \{0,180.0\}, \{0,180.0\}\}$	Define los pasos en los que aumentará o disminuirá el ángulo de rotación en grados hasta un máximo determinado por la regla <b>Rotation Range</b> .
Minimum Scale Factor	Punto flotante	1.0	$\{\{0.02,5.0\}\}$	Define el valor mínimo de escala que se le puede aplicar al objeto al distribuirlo.
Maximum Scale Factor	Punto flotante	1.0	$\{\{0.02,5.0\}\}$	Define el valor máximo de escala que se le puede aplicar al objeto al distribuirlo.
Appear Weight	Punto flotante	1.0	$\{\{0.0,1.0\}\}$	Define el peso del objeto, que se tendrá en cuenta a la hora de elegir entre los objetos a distribuir. Cuanto mayor sea el peso, mayor será la probabilidad de aparición del objeto en el entorno.
Adjust to normal	Punto flotante	1.0	$\{\{0.0,1.0\}\}$	Define el porcentaje de ajuste a la normal que se aplicará a todos los objetos de la escena, sin diferenciar el tipo.

**Tabla 4.1:** Reglas sobre las propiedades del objeto. La primera columna indica los nombres de las variables de cada regla.

#### 4.6.3. Reglas sobre la limitación de la distribución

Si bien los objetos, individualmente, pueden modificarse para obtener un resultado variado, es necesario tener en cuenta la manera en la que interactúan entre ellos. Para ello, se han definido tres reglas adicionales. Los detalles más puntuales de cada una de ellas se aclaran en la Tabla 4.2.

Regla	Tipo	Por defecto	Límites	Descripción
Distance Between Items	Punto flotante	0.0	[{0.0,100.0}]	Define la distancia mínima que debe tener el centro del objeto con el centro del resto de objetos distribuidos.
Don't Allow Overlap	Booleano	True	[{False, True}]	Establece si se permite o no que el objeto, al ser distribuido, se superponga con el resto de objetos de la distribución
Use Box	Booleano	False	[{False, True}]	Establece si la comprobación de la superposición será mediante la comparación de cajas delimitadoras o, por el contrario, mediante la comparación de esferas delimitadoras.

**Tabla 4.2:** Reglas sobre la limitación de la distribución.

En primer lugar, se dispone de una regla que define la distancia mínima que debe haber entre el centro del objeto cuya regla se modifique, y el centro de cualquier otro objeto que se coloque mediante el uso de la extensión de *Blender* (o potencialmente cualquier otro programa de gráficos tridimensionales). Si la distancia mínima provoca que la distribución sea imposible debido a la limitación de espacio, no se producirá. De manera similar, se llegó a plantear una regla que restringiese la distancia máxima que debe haber entre objetos, pero se decidió que carecía de suficiente relevancia en la primera versión del trabajo. Sin embargo, para versiones más avanzadas, podría ser de utilidad, así que se ha decidido tenerla en cuenta como trabajo futuro.

En segundo lugar, se ha creado una regla que determina qué objetos permiten que el resto de elementos a distribuir se puedan superponer con ellos. Esta regla utiliza las *esferas delimitadoras* de cada objeto (Figura 4.5). Las esferas delimitadoras son, como su nombre indica, esferas cuyo radio coincide con el extremo del objeto más alejado del centro. La esfera es necesaria para simplificar la detección de la superposición, puesto que si no se utilizase, el cálculo que determina si algún vértice de la malla interseca con el de otro objeto requeriría mucha más complejidad. De esta manera, estos cuerpos de revolución abarcan el volumen completo de cada elemento, pudiendo así comparar de manera sencilla si éstos se superponen entre ellos.

Este planteamiento a la hora de calcular las superposiciones se origina, sobre todo, de

la idea de evitar que las rotaciones aplicadas a cada objeto (véase Sección 4.6.2) supongan un problema. Siendo una esfera lo que se tiene en cuenta, una rotación en cualquier eje no supondrá que haya que modificar también el polígono delimitador de la misma manera. Esta es la razón por la que se desechó la idea original, utilizar cajas delimitadoras (Figura 4.5). Cuando a un objeto se le aplica una rotación, es preciso que el programa vuelva a calcular cuales son los vértices que conforman el extremo de la malla del objeto. Por ello, el uso de una caja delimitadora requiere que se tenga en cuenta, tanto cada vértice de la malla, como el centro para, acto seguido, aplicar las rotaciones pertinentes. Estos datos son propios del programa de modelado 3D y, al ser la distribución independiente del mismo, no sería sencillo obtener el nuevo volumen delimitador. Por lo tanto, se ha decidido que tratar de utilizar este método más óptimo queda como trabajo que se completará en el futuro.

Sin embargo, es importante destacar que se ha mantenido la posibilidad de utilizar cajas delimitadoras, debido a que, si no se pretenden realizar rotaciones, se trata de un poliedro que es capaz de encapsular con mayor eficiencia el volumen del objeto a distribuir. En el caso de objetos cuyo tamaño en un eje es mucho mayor que en los otros 2, una esfera delimitadora tiene en cuenta el volumen del objeto con un radio equivalente al tamaño de un vértice extremo de ese eje. Por ello, una gran parte de la esfera cubrirá cierto espacio que el elemento no ocupa, impidiendo la superposición de otros objetos con un espacio que está vacío y, por consiguiente, debería poder ser ocupado. Esta es la razón por la que se permite, mediante una tercera regla, delegar en el usuario la responsabilidad de decidir qué polígono delimitador tendrá en cuenta cada objeto.

#### 4.6.4. Implementación de las reglas

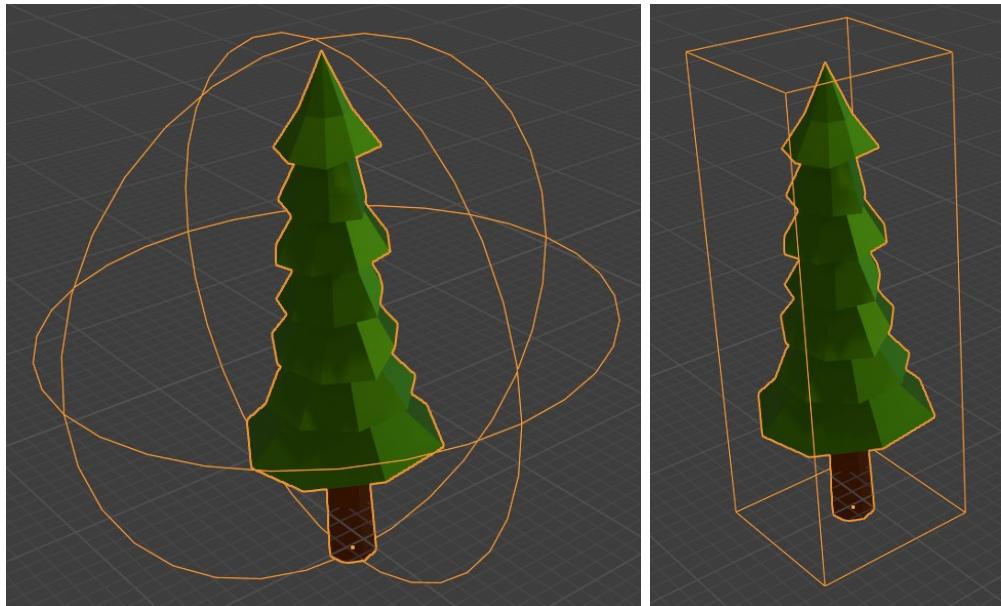
Para implementar el sistema de reglas, se ha partido de la clase `ItemRules`, mencionada en la Sección 3.2.2. Esta clase contiene la información de cada una de las reglas de los objetos y habrá una instancia de la misma por cada objeto diferente a distribuir. Las reglas guardadas en esta clase serán especificadas por el usuario desde la interfaz gráfica de la extensión (véase Sección 5.3).

Una vez definidas las reglas de cada elemento, éstas son utilizadas por la clase `DistributionProblem` (detallada en la Sección 4.4). Esta clase hace uso del método `CheckRestrictions`, que se encarga de determinar, regla a regla, si el objeto posicionado por una acción, cumple todas las restricciones que le han sido impuestas. Este método es llamado durante el ciclo de distribución por la función `Acciones`. Si el elemento cumple dichas restricciones, éste es

guardado en la lista de *acciones* (véase Sección 4.2) y pasará a formar parte de la solución. En caso de que las reglas no sean cumplidas, se descarta esta *acción* como parte de la solución.

De manera previa a la llamada de la función `CheckRestrictions`, se seleccionará el objeto que podrá pasar a formar parte de la solución. Esto se realizará teniendo en cuenta el peso de aparición de cada elemento (no confundir con *Weight Paint*), de manera que los elementos con mayor peso tendrán más probabilidad de ser seleccionados. Tras ello, se llama a la función. En primer lugar, se define el factor de la escala (Sección 4.6.2) y, sobre el objeto con este factor añadido, se comprueban todas las restricciones mencionadas en secciones previas. Para llevar esto a cabo, se itera sobre la lista de acciones que conforman la solución y se comprueba que todas ellas permiten la colocación de dicho elemento.

Si el elemento cumple las restricciones, se le define, o no, la variación sobre la rotación acorde a la configuración de dicha regla. Una vez se completa el ciclo de ejecución de *AI-MA*, tanto las rotaciones como la escala son aplicadas a cada objeto en el momento de su distribución, mediante el programa 3D en cuestión.



**Figura 4.5:** Volúmenes Delimitadores en Blender: esfera a la izquierda y caja a la derecha.

# Capítulo 5

## Integración de la herramienta en un programa 3D

Este capítulo se centrará en desarrollar la implementación de la extensión en el programa de modelado *Blender*. Se abordarán las razones por las que se ha elegido este programa, comenzando por explicar los operadores de *Blender* y la forma en la que se utilizan en este trabajo.

A continuación se hablará también de las herramientas y utilidades que han sido desarrolladas para agilizar el trabajo del usuario de la herramienta. Con relación a esto, se hablará de la interfaz de usuario, donde se detallará la organización de cada una de las herramientas y funcionalidades. Además, se expondrá el flujo de trabajo que se debe llevar para obtener buenos resultados al utilizar la extensión.

### 5.1. Implementación de operadores en *Blender*

Para incorporar los algoritmos que distribuirán los objetos a *Blender* se ha hecho uso de operadores. En *Blender*, un operador es una función o comando que realiza una acción específica dentro de la aplicación [51], generalmente ejecutada mediante un botón de la interfaz. Además de con botones, los operadores se pueden llamar desde la interfaz de usuario a través de menús y mediante el uso de atajos de teclado. Cada operador tiene un nombre único y se puede acceder a él a través de la *API* (definida en la Sección 1.4.1) de *Blender*. La *API* de *Blender* es una colección de módulos y funciones que permiten a los desarrolla-

dores y usuarios personalizar y automatizar la funcionalidad de *Blender*. Dichos operadores se pueden combinar en secuencias, llamadas “*macros*”, para realizar acciones complejas y repetitivas de manera más eficiente.

Los operadores se crean como clases en *Python* que heredan de `bpy.types.Operator`, conteniendo así una serie de atributos de identificación, lo que permite acceder a ellos en el resto del código del programa. De esta manera también se permite acceder a los métodos necesarios para ejecutar las acciones que pretenda el programador. El método más importante de un operador es “`execute`”, que es el que se ejecuta al interactuar con el operador mediante la interfaz asociada (la interfaz de usuario de *Blender* se explica en la Sección 5.3). Este método es redefinido por el programador para realizar una tarea concreta, como por ejemplo, llevar a cabo la distribución de objetos sobre una superficie. Además de esto, existe un método “`poll`”, que se encarga de determinar si el método anterior está, o no, habilitado en función del estado del entorno y contexto programa. Si no está habilitado, el elemento de la interfaz asociado al operador será bloqueado y no se podrá utilizar.

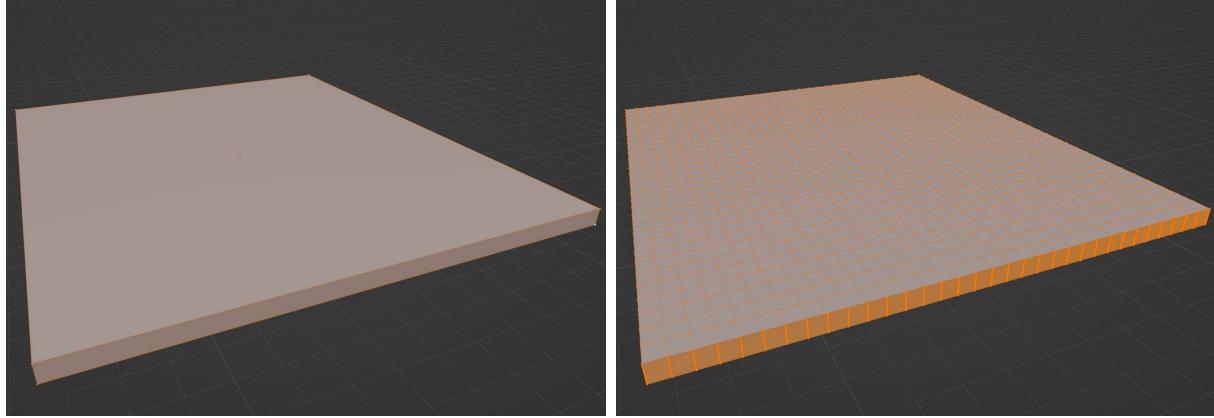
En el caso de este trabajo, existen una serie de operadores que, utilizando la información del escenario 3D, ejecutan el código de distribución que hace uso de los algoritmos explicados en la Sección 4.3. La estructura de estos operadores ha sido detallada en la Sección 3.2.1. La extensión hace uso del método “`poll`” para bloquear los operadores previamente mencionados mientras se usa la utilidad de pintado de vértices explicada en el siguiente apartado (Sección 5.2).

## 5.2. Utilidades y herramientas de la extensión

Para utilizar correctamente la extensión, se deben disponer una serie de elementos de manera que permitan la distribución de los objetos correctamente. *Blender* posee las utilidades que permiten colocar todo lo necesario para que la extensión pueda ser utilizable. Sin embargo, el programa no está acondicionado para que sea sencillo o intuitivo acceder a ellas. Por ello, se han añadido una serie de herramientas a la extensión que permiten acceder a estas utilidades de manera sencilla y maquinal.

En primer lugar, se debe capacitar la superficie objetivo, donde se repartirá cada objeto, con los ajustes necesarios para comenzar la distribución. Para que esta superficie cumpla su función, deberá estar dividida como mínimo en tantos vértices como objetos se pretenda distribuir. En *Blender* se pueden realizar secciones en una malla para crear superficies más

suaves y detalladas que se adaptan mejor a las texturas y a los materiales que se aplican. Esto se logra mediante la adición de más vértices, aristas y caras en la malla seleccionada, sin modificar el aspecto de la misma (Figura 5.1). En el caso de este trabajo, la adición de vértices es lo que permite obtener más puntos donde se colocarán los elementos distribuidos.



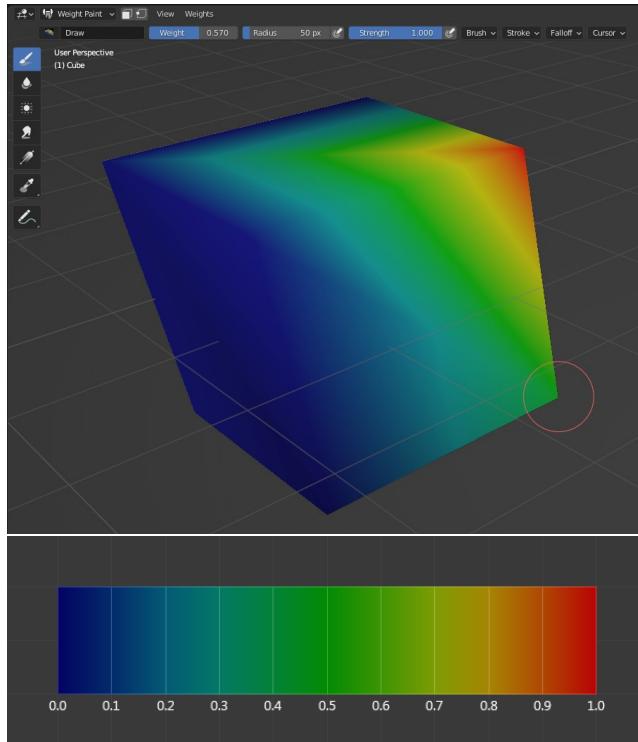
**Figura 5.1:** De izquierda a derecha: misma superficie con creciente cantidad de subdivisiones en *Blender*.

Al ser esta acción necesaria para el correcto funcionamiento de la extensión, se ha decidido añadir una herramienta a la interfaz de la extensión que permita realizarla de manera simple. Al igual que en la utilidad original de *Blender*, se permitirá al usuario seleccionar el número de subdivisiones (o cortes de la malla) que desee. Originalmente, si no se seleccionaba un número de subdivisiones, se hacían automáticamente de manera que la distancia entre vértices fuera igual al extremo más largo del objeto entre la anchura y la profundidad. Dicha funcionalidad tenía en cuenta que la altura no debería afectar al número de subdivisiones, ya que en una distribución sobre una superficie horizontal, solo se tendrían en cuenta los ejes *X* e *Y* (en *Blender*). Este automatismo quedó obsoleto cuando, más adelante, se decidió añadir la regla de superposición (véase Sección 4.6.2), así como la posible distribución de múltiples elementos (véase Sección 4.4). Al permitir superposición entre objetos, no es óptimo que la subdivisión ignore esta posibilidad, creando puntos de manera que las superposiciones sean imposibles. Por otro lado, si se distribuyen múltiples elementos, si bien se podría seleccionar el lado más largo de todos ellos, podría no resultar óptimo para otros objetos de diferente forma y tamaño.

Esta utilidad también fue implementada teniendo en mente que el usuario podría no querer modificar la malla de su superficie. En este caso, aplicar subdivisiones a una malla supondría un problema. Por ello, la herramienta que se ha desarrollado sigue ciertos pasos

para solucionar este problema. Antes de todo, se duplica la malla de la superficie objetivo. Sobre esa malla duplicada, se aplican las subdivisiones pertinentes, cuyo número será decidido por el usuario. Una vez se haya completado la distribución de objetos, la malla generada es eliminada, dejando la superficie original de la misma manera en la que se encontraba inicialmente.

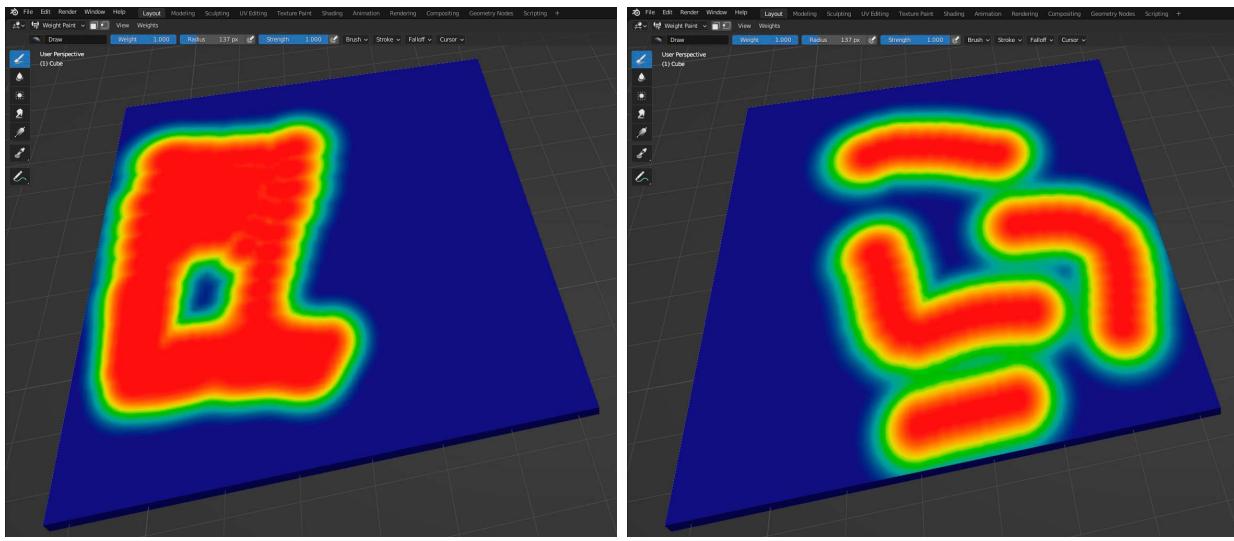
La segunda utilidad que se va a tratar es la que permite modificar el peso de los vértices. El peso de los vértices se muestra como un valor numérico entre 0 y 1 en *Blender*, donde 0 significa que no se distribuirá ningún objeto sobre él y 1 significa que el vértice se tendrá en cuenta para la distribución. Los valores intermedios representan un peso parcial, que podrá ser tenido en cuenta para la distribución o no, según considere el usuario. Visualmente, los valores de peso de un vértice se presentan como un gradiente de color, donde 0 es azul oscuro y 1 es rojo (Figura 5.2).



**Figura 5.2:** Gradiente de color indicativo del peso de un vértice en *Blender* (imagen obtenida de “The Blender 3.5 Manual” por el equipo de documentación de *Blender*, bajo la licencia CC-BY-SA v4.0).

Para facilitar la gestión del peso de los vértices, se ha creado una segunda utilidad. Esta utilidad permite al usuario acceder rápidamente a la interfaz de pintado de vértices, sobre

la que se pueden aplicar una serie de comandos. En primer lugar, se pueden definir varios perfiles, en los que se pueden almacenar, para la misma superficie, diferentes configuraciones de peso de los vértices (Figura 5.3). Esta función se diseñó para permitir que el usuario seleccionase diferentes regiones para distribuir por ellas grupos de elementos distintos entre sí. Aparte de esto, la herramienta cuenta con dos añadidos para facilitar la interacción del usuario. Antes de todo, posee una opción que permite invertir los pesos de todos los vértices. En segundo lugar, se proporciona una opción para aplicar a todos los vértices de la superficie un peso específico. Estas funciones se plantean para casos en los que el usuario no necesite seleccionar áreas de la superficie de manera muy concreta.



**Figura 5.3:** Interfaz de pintado de vértices: misma superficie con diferentes perfiles de configuración de pesos de vértices.

A continuación, se va a hablar de las utilidades que actúan sobre los elementos a distribuir. El usuario tendrá acceso en todo momento a los objetos que se van a posicionar en la superficie. Sin embargo, para evitar un flujo de trabajo lento, se ha facilitado la posibilidad de seleccionar cada objeto por separado en el visualizador 3D o, en su defecto, seleccionar todos ellos al mismo tiempo. El visualizador 3D es el área de *Blender* que muestra el escenario tridimensional sobre el que se encontrarán los volúmenes colocados por el usuario [52]. Asimismo, se permite que el usuario pueda modificar el orden de los objetos en la lista, además de borrar cada uno de ellos por separado o todos al mismo tiempo.

Por último, existen dos utilidades relacionadas con la superficie objetivo. Ambas se relacionan con el uso que hace la extensión de las normales (según están definidas en la

Sección 4.6.2). La primera permite cambiar el modo de visualización de las mallas de la escena, de forma que sea observable la orientación de cada cara de las mismas. De esta manera, se puede detectar si la superficie objetivo posee alguna cara invertida. Si éste es el caso, cualquier objeto que se coloque encima, y se ajuste a las normales, estaría en la posición inversa a la esperada. En relación a esto, se ha implementado la segunda utilidad, que permite calcular de nuevo la orientación de las caras de la superficie para corregir los defectos de la malla.

### 5.3. Disposición de la interfaz gráfica de usuario

Para exponer cómo se ha desarrollado la interfaz gráfica de usuario de la extensión, se debe hablar de cómo funcionan los paneles de *Blender* [52]. En *Blender*, los paneles son las áreas de la interfaz de usuario donde se encuentran las herramientas, los ajustes y los menús que se utilizan para trabajar con objetos 3D. Se organizan en diferentes zonas, como la ventana principal, el panel de propiedades, el editor de nodos, etc. Cada panel se compone de diferentes elementos, como botones, menús desplegables, campos de entrada de datos y ventanas de vista previa. Estos elementos se organizan en pestañas para ayudar a organizar la información y las herramientas de forma lógica. Este es el caso de las pestañas que conforman las funcionalidades de la extensión. Además, los paneles se pueden personalizar y modificar según las necesidades del usuario. Por ejemplo, se puede alterar el tamaño de los paneles o ajustar la disposición de los elementos dentro de cada panel en cuanto a la posición.

La extensión hace uso de varios paneles que se podrán desplegar para dar paso a cada funcionalidad de la extensión. El primer panel se presenta con el nombre “*Object To Distribute*”, y presenta los servicios más importantes. Todos los campos y botones del panel serán indicados con un número entre paréntesis, que indica la correspondiente referencia en la Figura 5.4. Las funciones más importantes del mismo son los tres botones de distribución, colocados al principio del panel. Estos botones ejecutarán el operador de *Blender* asociado a la distribución de objetos (Sección 4.3).

El primer botón del panel (1) ejecuta la distribución de un objeto únicamente. Este objeto será el que se encuentre seleccionado en el cuadro que se encuentra inmediatamente debajo de los botones. El segundo botón (2) ejecuta una distribución múltiple, la cual reparte por la superficie objetivo todos los elementos que se encuentran en el cuadro previamente



**Figura 5.4:** Panel de objetos a distribuir, superficie objetivo y sus funcionalidades. Los números en circulo son indicativos para referirse a ellos.

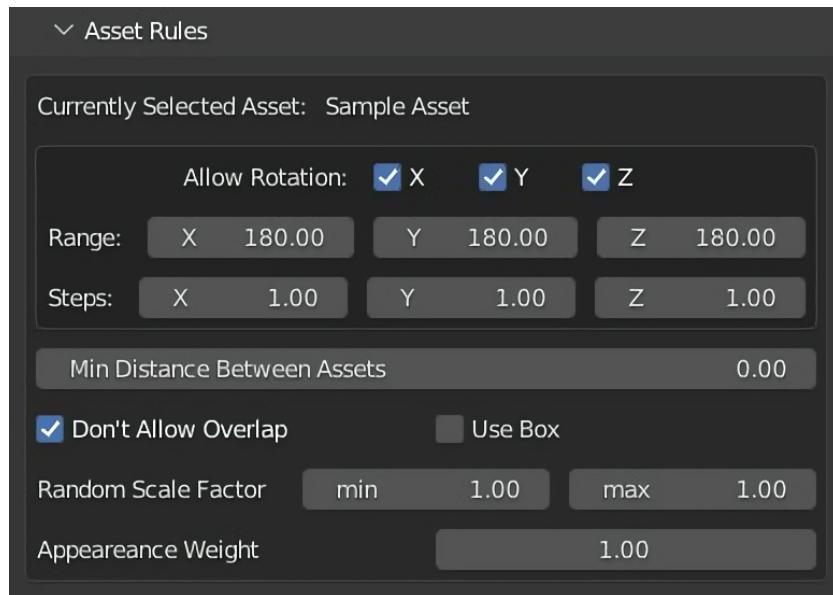
mencionado. Estos botones no se tendrán en cuenta si ya se había producido una distribución previamente (excepto si se han tenido en cuenta los objetos que son parte de la solución), realizando una nueva distribución de los mismos objetos con cada ejecución. El último de los botones (3) eliminará todos los objetos de la colección seleccionada.

El cuadro que aparece a continuación (4), como ya se ha comentado, contiene todos los elementos que se van a distribuir. Los botones marcados con los signos + y - (5), pueden ser utilizados para añadir o eliminar elementos a dicho cuadro. Todos los botones que aparecen a continuación (6) son los que ejecutan las funcionalidades explicadas en la sección anterior (penúltimo párrafo de la Sección 5.2).

Este panel también recoge información sobre la superficie objetivo, donde se distribuirán los objetos. Respecto a este apartado, el componente más importante de la interfaz es el campo que contiene el objeto que hará las veces de superficie objetivo (7). A continuación este campo, se encuentran los botones que ejecutan varias de las utilidades que se desarrollan

en la sección anterior (Sección 5.2). En primer lugar, se encuentran las utilidades referentes a la gestión de normales y orientación de las caras y vértices (8). En segundo lugar, se encuentra la herramienta de pintado de vértices (9) que, como se explica en la sección anterior, permite seleccionar fácilmente los vértices sobre los que se distribuirán los objetos.

A continuación se hablará del panel de reglas de los objetos (Figura 5.5). Este panel contiene todos los campos con diferentes parámetros que se desarrollan en el Capítulo 4. Encabezando el panel, aparece un texto indicativo de cuál de los objetos, entre los que se encuentran en el cuadro del panel anterior (4), está seleccionado. Si no hay ninguno seleccionado, no aparecerán las reglas, ya que son individuales de cada uno de ellos. La funcionalidad de cada regla se detalla en las secciones 4.6.2 y 4.6.3.



**Figura 5.5:** Panel de Reglas de los objetos.

El panel que se trata a continuación es el que se encarga de gestionar todos los parámetros que han tenerse en cuenta para efectuar la distribución (Figura 5.6). A diferencia del panel anterior, que trataba los parámetros y reglas específicos de los objetos, este panel cubre todos aquellos componentes que configuran la distribución en su totalidad. El primer campo que encontramos es la selección del nombre que tendrá la colección donde se generan todos los objetos. En *Blender*, una colección es un contenedor que se utiliza para organizar objetos en la escena 3D. Los objetos en *Blender* se agrupan en colecciones, y cada objeto pertenece a una o varias colecciones. Dichas colecciones pueden contener cualquier tipo de elemento, como mallas, cámaras, luces y vacíos. Además, las colecciones se pueden anidar, lo que

significa que una de ellas puede contener otras colecciones. Si no se configura la colección a la que pertenecerán los objetos seleccionados, no se permitirá ejecutar la distribución. Adicionalmente, en el caso de que se decida distribuir un conjunto de objetos diferente al que se haya distribuido inicialmente, se generará una colección con un nombre idéntico a la anterior, seguido de un identificador numérico.

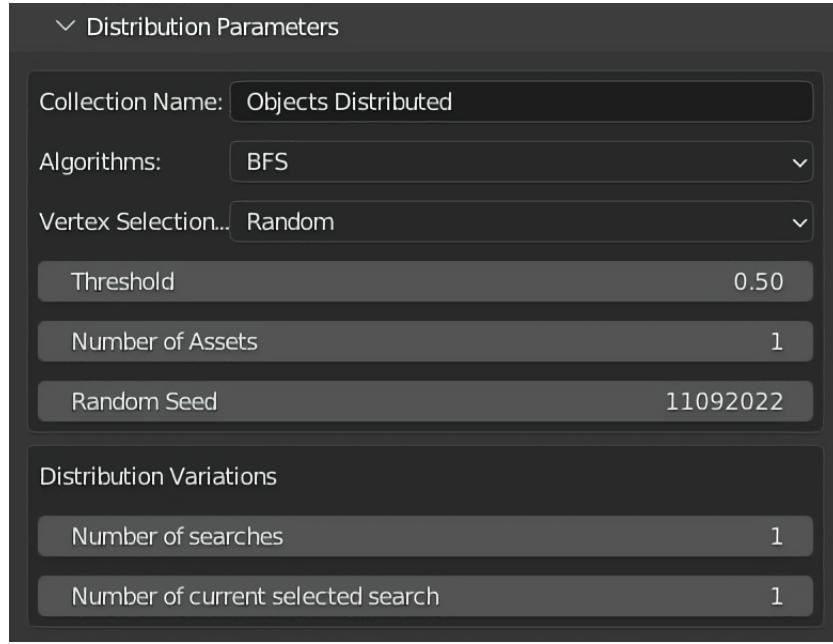
La siguiente opción del panel permite seleccionar qué algoritmo de distribución se va a utilizar (véase Sección 4.3). Se permite al usuario elegir entre los tres disponibles: “*Búsqueda del Primero Mejor*”, “*Escalada Simple*” y “*Temple Simulado*” (véase Sección 4.3). La opción a continuación sirve para seleccionar el peso mínimo que debe tener un vértice para poder colocar un objeto sobre él. Por otro lado, uno de los elementos más importantes del panel es el número de objetos. Esta sección es la que define la cantidad de elementos que se repartirán por la superficie. Si, porque las reglas son muy restrictivas, o porque se ha seleccionado un elevado número de elementos, la distribución resulta imposible, no se efectuará y se indicará al usuario con una advertencia.

El siguiente elemento del panel es la semilla que se usa en la distribución aleatoria. En el contexto de la generación de números aleatorios, una semilla es un valor inicial que se utiliza para iniciar un algoritmo de generación de números aleatorios (véase Sección 2.1.1). La semilla determina el conjunto de números que se generarán a partir del algoritmo, y si se utiliza la misma semilla, se generará el mismo conjunto de números aleatorios cada vez. De esta manera, se puede replicar una distribución previa usando esta semilla.

El siguiente cuadro presenta dos opciones. La primera de ellas es el número de búsquedas que se quieren efectuar. Si se configura el número de búsquedas, se realizarán tantas distribuciones como se haya establecido si el algoritmo lo permite. La siguiente opción permite alternar entre las distribuciones que se hayan realizado, una vez han sido realizadas.

El penúltimo panel que conforma la interfaz de usuario de la extensión sigue una estructura idéntica al cuadro de selección de objetos a distribuir (Figura 5.7). Este panel permite añadir objetos de la escena los cuales se tendrán en cuenta en la distribución. Esto implica que se cuenten entre el total de objetos que se van a distribuir, además de ser considerados cuando se deban aplicar las reglas del resto de elementos.

Por último, se creó un panel que aplicase la funcionalidad de la herramienta de subdivisión explicada en la (Sección 5.2). El panel permite al usuario decidir si pretende aplicar una subdivisión a la superficie objetivo, y el número de cortes que dicha subdivisión tendrá.

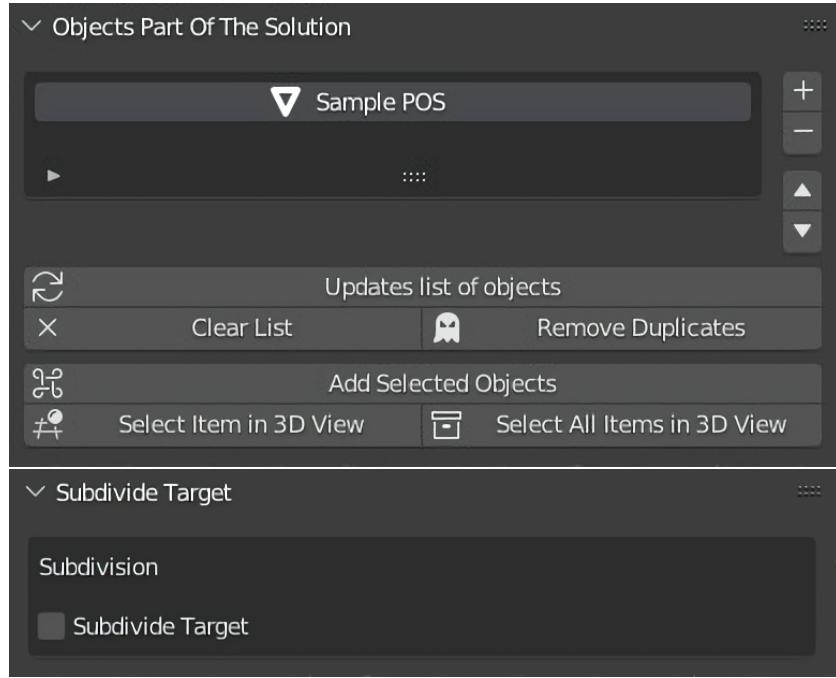


**Figura 5.6:** Panel de parámetros de la distribución.

## 5.4. Instalación de la herramienta en el programa 3D

Para instalar cualquier extensión dentro de programa Blender, basta con comprimir en un archivo *.zip* el contenido del código fuente de *Python*, e instalarlo en la sección de *Add-ons* del menú de preferencias. Sin embargo, este proyecto utiliza los módulos de *AIMA* para su correcto funcionamiento, los cuáles no vienen incorporados en las librerías base de *Python*. Esto implica instalar manualmente dicho módulo desde la distribución de *Python* que viene junto con Blender.

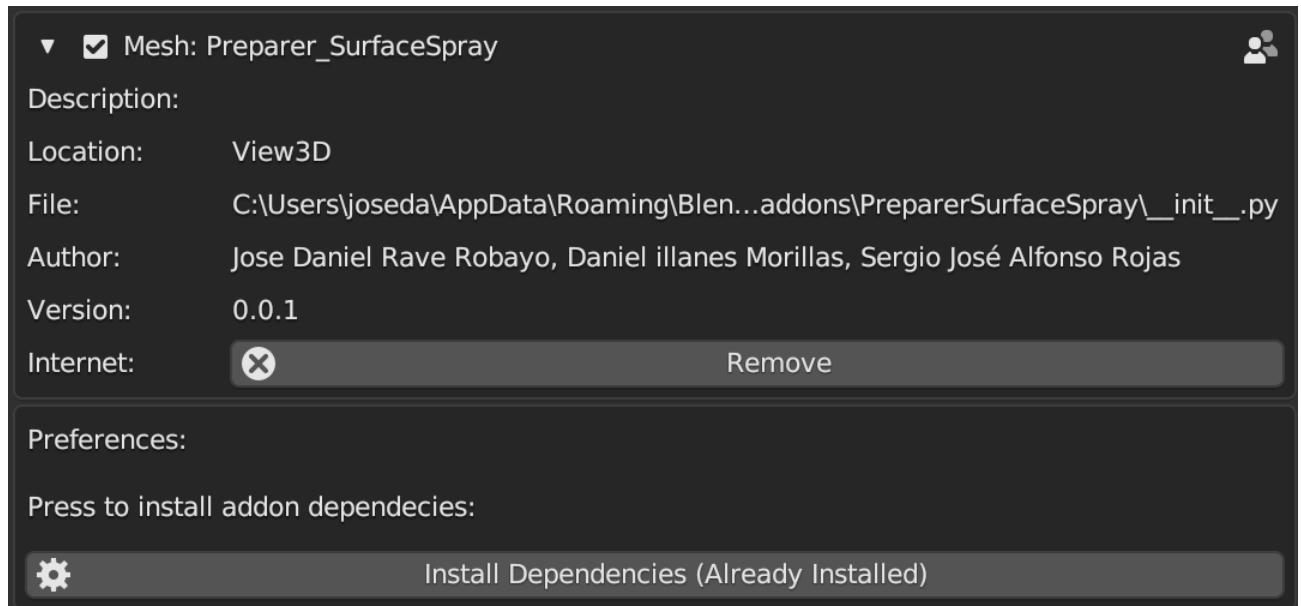
Esta tarea, incluso para usuarios con experiencia en programación, puede llegar a ser una tarea que complica el proceso de la instalación, pues hace falta acceder a los directorios internos de *Blender*, localizar el ejecutable de *Python*, y proceder con el comando para instalar *AIMA*. Originalmente se diseñó un documento que contenía una serie de comandos a seguir e instrucciones para la instalación de *AIMA*. Pero esto aún requería ciertos conocimientos de programación e instalación de dependencias. Por lo tanto, se planteó que la propia extensión fuese capaz de instalar automáticamente dichas librerías a la hora de ser registrada dentro del programa. No obstante, Blender no permite el registro de ficheros que contengan módulos importados de origen desconocido. Así que se propuso extraer esta instalación automática a otra extensión, la cual no almacenara ningún fichero que importe



**Figura 5.7:** Panel de parámetros de los objetos que se interpretan como parte de la solución (arriba). Panel de subdivisión (abajo).

módulos de *AIMA*, permitiendo así su registro en Blender y la consiguiente instalación de las dependencias. En la Figura 5.8 se pueden ver el panel de instalación de extensiones de Blender, en el que está incorporado dicha extensión.

Tanto en el repositorio del proyecto como en el Apéndice Notas de instalación del repositorio se encuentra una descripción detallada de cómo instalar ambas extensiones. Además, la evaluación con usuarios (véase Sección 6.2.2) permitió saber qué tan intuitiva era esta descripción, y en base a las sugerencias e inconvenientes que presentaban los usuarios, se hicieron los cambios correspondientes. Cabe resaltar que en la misma descripción se encuentra el documento original de instalación de *AIMA* previamente mencionado.



**Figura 5.8:** Panel de instalación de Blender. La extensión que instala las dependencias se llama *PreparerSurfaceSpray*, siendo *SurfaceSpray* la extensión principal de este trabajo. El botón “*Install Depedencies*” se encargará de realizar dicho proceso.

# Capítulo 6

## Evaluación

*“Tropezar también es avanzar.”*

---

Orisa, Overwatch.

Habiendo comprobado que la extensión es operativa y se podía instalar en múltiples dispositivos, se comenzó el diseño de los diferentes experimentos que compondrían la evaluación del programa. De esta manera se procedió a estudiar si el trabajo cumple con los objetivos planteados originalmente y confirma las hipótesis presentadas en la Sección 1.2. Además, mediante los resultados se reflejará si el programa es intuitivo y claro a la hora de ser usado.

Para llevar a cabo la fase de experimentación, se han diseñado varios planes de pruebas, ya que conforme se realizaban las pruebas, éstos sufrían cambios cruciales, con el objetivo de obtener resultados de mejor calidad.

### 6.1. Método de evaluación

El plan de evaluación pretende llevar a cabo dos experimentos diferentes con el objetivo de extraer la máxima información de algunos aspectos de la herramienta, y poder así realizar un análisis. El segundo experimento será diseñado teniendo en cuenta qué componentes del anterior no cumplen el objetivo de los mismos o suponen un contratiempo que dificulta la interacción del usuario. Los usuarios que tomen parte en el primer experimento serán diferentes a los que participen en el proceso de evaluación del segundo.

La evaluación tendrá como objetivo comprobar si los resultados confirman la hipótesis planteada en la Sección 1.2. Para demostrar que se cumple, los experimentos se centrarán en medir la rapidez con la que los usuarios consiguen completar las pruebas. Sin embargo, aunque el objetivo principal sea la medición del tiempo, también se pretende monitorizar todos los errores que aparezcan, así como las imperfecciones que ralenticen el flujo de trabajo del usuario.

Las pruebas tienen una duración máxima de 50 minutos por usuario y se realizarán de forma remota. Cada usuario recibirá las instrucciones de un solo evaluador, que serán especificadas en un documento. El evaluador se encargará de dar la información mínima necesaria, y evitar actuar de manera que el resultado de la prueba quede invalidado. Mientras el usuario realiza las pruebas, el evaluador se encargará de tomar nota de los comportamientos y reacciones del evaluado, así como de resultados inesperados para su posterior análisis. Todas las sesiones de pruebas serán grabadas para su posterior análisis (ver Apéndice Enlaces relacionados con el proyecto) .

## 6.2. Proceso de evaluación

El proceso de evaluación está compuesto por dos experimentos. El primero será enfocado en un prototipo diseñado para poder realizar unas observaciones iniciales sobre el funcionamiento de la herramienta desarrollada. A partir de este experimento, se formulará el segundo, que tendrá en cuenta el análisis de la fase previa e involucrará a un mayor número de usuarios. De esta manera, se espera obtener información que permita validar las hipótesis que se plantean en la la Sección 1.2. El número total de usuarios que han sido evaluados es de 15 personas.

Es importante resaltar que para realizar estos experimentos, se ha simplificado la interacción de los usuarios respecto al apartado de selección de algoritmos (ver Secciones 4.3 y 5.3). En esta fase, se indicará a los usuarios que utilicen exclusivamente uno de ellos, en concreto *Búsqueda del Primero Mejor*, pues este no sólo garantiza posicionar todos los objetos, sino que también consigue múltiples soluciones. Se ha tomado esta decisión debido a que el foco de los experimentos se ha puesto en obtener datos sobre la satisfacción de los usuarios y la velocidad de su flujo de trabajo.

### 6.2.1. Primer experimento

En primer lugar, se debe conocer el perfil del usuario. Es decir, a qué se dedica profesionalmente o cuáles son sus principales intereses, cuán familiarizado está con programas de modelado tridimensional, y el nivel de experiencia que tiene, concretamente, con el programa *Blender*. Se utiliza para la medición una escala de 1 a 5, siendo 1 nivel principiante, y 5, experto. Para ello, se pide llenar un breve formulario realizado en *Google Forms* (visitar Sección C.1 del Apéndice Documentos del proceso de evaluación) que reúne diferentes preguntas necesarias para conocer estos datos de los encuestados. Además de esto, se categorizará al usuario según edad y género, pues es de utilidad organizar los resultados por grupos. Con este experimento se pretende conocer si la extensión es intuitiva para los usuarios y si, mediante su uso, se puede formular un segundo experimento que se enfoque en estudiar si se cumplen las hipótesis planteadas en la Sección 1.2.

La primera prueba consiste en pedir a la persona evaluada que, en base a una imagen como referencia, intente reproducir dicha imagen partiendo de un proyecto plantilla de *Blender* creado de antemano. Esta plantilla parte de la superficie donde se repartirán los objetos, que también se incluyen en ella. Esta parte del experimento no utilizará la extensión, de manera que se pueda monitorizar el desempeño del usuario en este contexto. El tiempo que el usuario tarde en recrear la escena es cronometrado y registrado para contrastar en un futuro. El escenario se muestra en la Figura 6.1.

Una vez finalizada la primera fase, se procede con la instalación de la extensión. Se proporciona el enlace al repositorio del proyecto y se da libertad para que, siguiendo las instrucciones de instalación incluidas en el proyecto, descargue e instale el mismo en su copia de *Blender*. En caso de ser necesario para continuar, se podrá intervenir para ayudar al evaluado. Se monitorizará de esta manera qué partes de las instrucciones de instalación resultan complicadas de seguir.

A continuación, una vez se ha instalado con éxito la extensión, se facilita al usuario una nueva escena en la que tendrá una mayor libertad para aprender a manejarla. Se espera que revisen y prueben todas las opciones y funcionalidades posibles, aunque centrando especial atención en las principales funciones de la herramienta. Para lograr esto se facilita un pequeño memo con instrucciones básicas de uso (Figura 6.2). En esta segunda prueba, se evaluará qué opciones, sin contar las obligatorias, son utilizadas por el usuario y cuales son ignoradas.

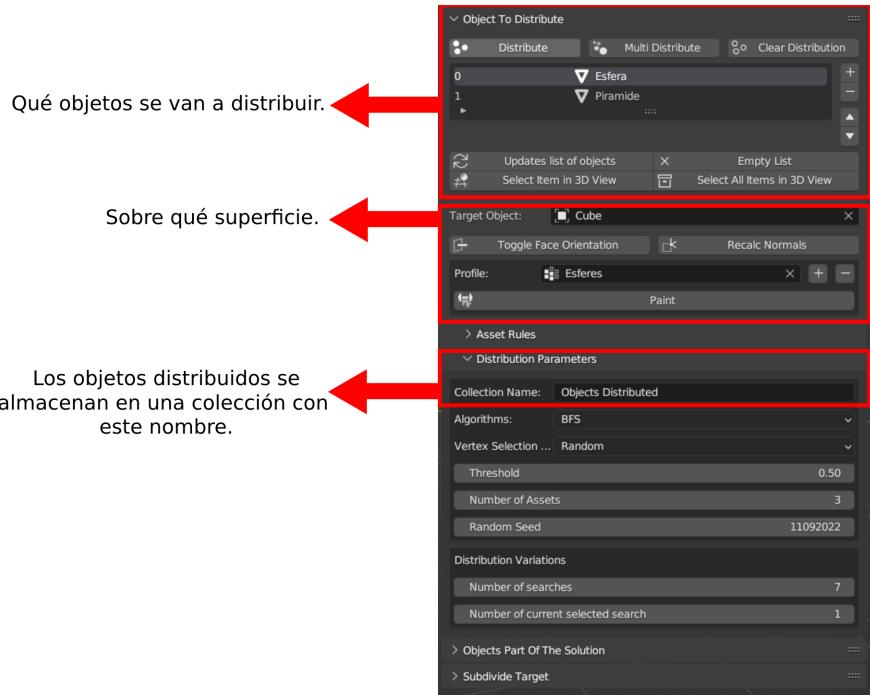


**Figura 6.1:** Escenario de ejemplo, utilizado como objetivo para las personas evaluadas.

Los principales puntos a cubrir por el usuario serán:

- Selección de objeto objetivo sobre el que distribuir.
- Selección de objeto/objetos a distribuir.
- Opción de distribución múltiple.
- Reglas individuales de objeto.
- Reglas individuales dentro de la selección múltiple.
- Pintado de vértices para distribución selectiva.
- Cambio de nombre de la colección que almacena el resultado.

Tras un corto periodo de tiempo para permitir al usuario adaptarse a la herramienta en una escena libre, se comenzará una explicación más detallada de todas las funcionalidades



**Figura 6.2:** Memorándum con instrucciones simples de uso de la extensión.

de la extensión. Durante la explicación, se exemplificará cada apartado de la extensión, de manera que el usuario se acostumbre a trabajar utilizándola.

La última prueba del primer experimento consiste en pedir al usuario que genere un escenario utilizando la extensión. El escenario será similar al de la primera prueba, pero contará con un número más amplio de elementos repartidos por el escenario. Al igual que con la segunda prueba, el tiempo que el usuario tarde en recrear la escena es cronometrado y registrado para contrastarlo. Además, en esta última prueba también se les presenta una imagen, similar a la primera, pero con una mayor cantidad y variación de objetos. (Figura 6.3).

Por ultimo, se entrevistará a los usuarios. En esta corta entrevista, se les plantearán las siguientes tres cuestiones:

- “¿De qué manera ha cambiado la extensión tu experiencia en la creación del escenario con respecto a sin ella?”
- “¿Si se diera el caso, la usarías en un futuro? ¿Para qué?”
- “¿Has entendido bien la interfaz y sus funcionalidades? Si hay algo que no has enten-



**Figura 6.3:** Escenario de ejemplo para las pruebas utilizando la extensión.

dido, ¿qué es?”

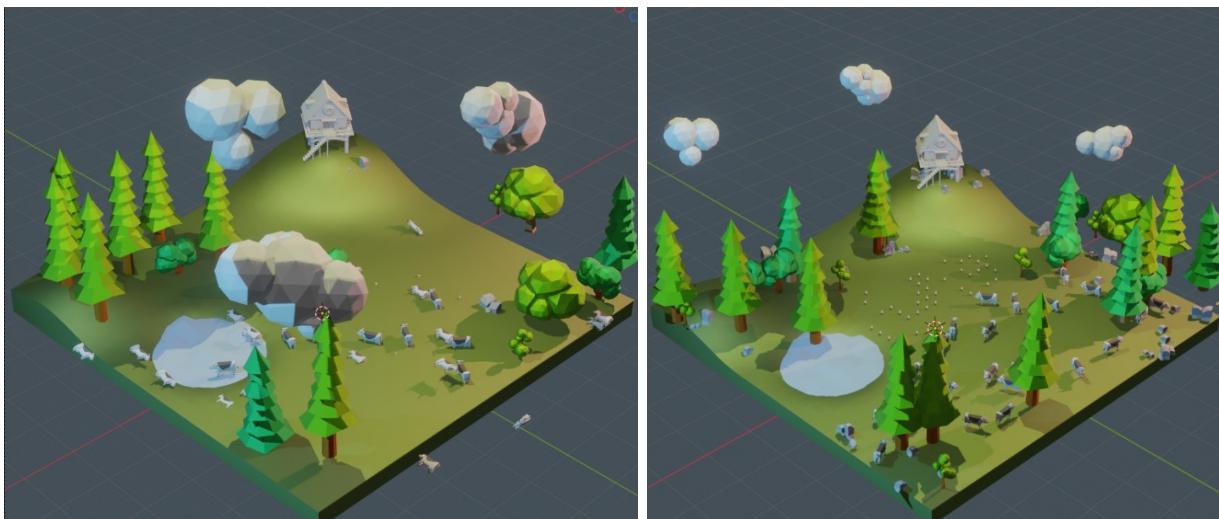
El primer experimento fue llevado a cabo con 5 usuarios, 3 mujeres y 2 hombres, residentes en ese momento en Madrid, España. Los usuarios tenían una edad de 21 años de media y eran estudiantes del *Grado en Desarrollo de Videojuegos*, en la *Universidad Complutense de Madrid*. Los 5 poseían un nivel medio en el manejo del programa *Blender*, lo que en la escala presentada de 1 a 5 se encuentra en una media de 2,4, con una desviación estándar de 0,49.

### Resultados del primer experimento

Tras seguir las pruebas que conforman el primer experimento, se obtuvieron una serie de resultados. Entre los 5 usuarios, la media de tiempo que se tardó en completar la primera prueba es de 14 minutos y 8 segundos, con una desviación estándar de 4 minutos y 18 segundos. Los escenarios resultantes de esta primera prueba seguían una estructura similar al que se presenta en la Figura 6.4. Si bien se indicó a los usuarios que no debían replicar

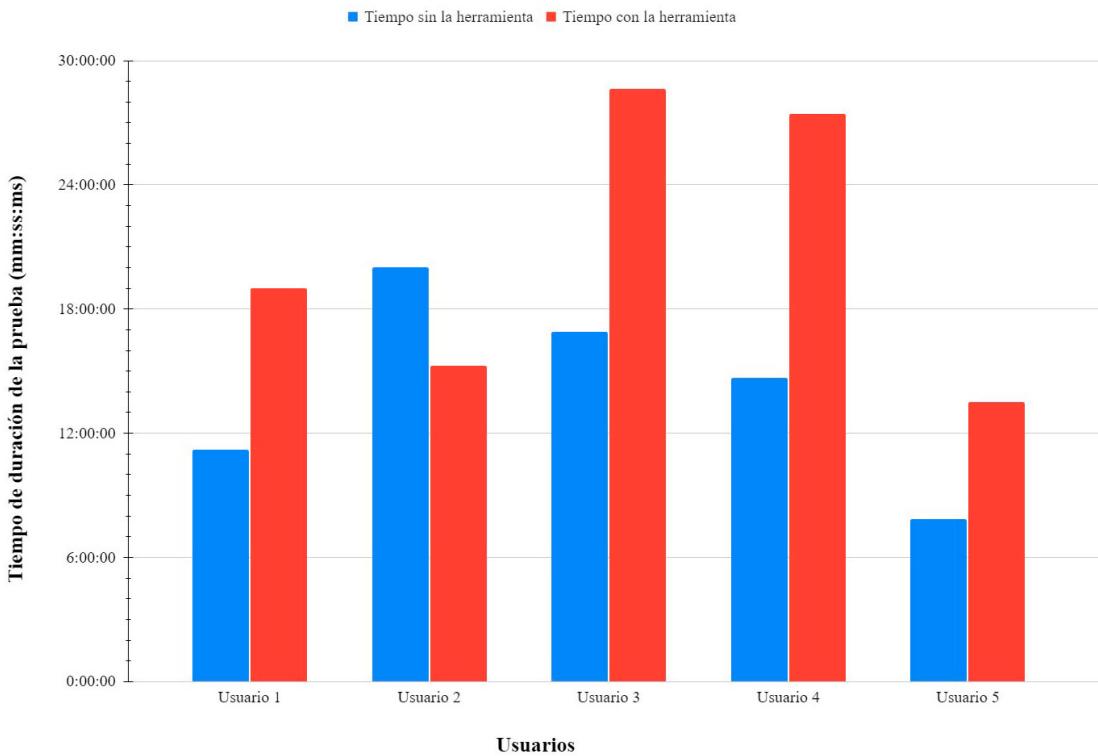
el escenario objeto por objeto, sino crear un entorno similar, los usuarios no seguían esta indicación. Para la instalación, se obtuvieron datos sobre el desempeño de los usuarios. El 40 % de ellos fueron incapaces de completar la instalación sin ayuda de los evaluadores, mientras que el 60 % restante sólo tuvo problemas mínimos de comprensión.

Durante la segunda prueba todos los usuarios entendieron los controles más básicos de la extensión. De forma común, ninguno de los usuarios trató de explorar en gran medida los elementos de la interfaz que no se les había explicado (todos menos los indicados en la Figura 6.2). El 40 % de los usuarios no acabaron de comprender cómo manejar algunas opciones de la herramienta de pintado de vértices. Además, los usuarios no manejan en una primera instancia el establecimiento de colecciones en *Blender* (véase Sección 5.3). La última de las opciones que resultaron poco comprensibles para los usuarios fue la regla que define la cantidad de grados que aumenta o disminuye la rotación del objeto (explicada en la Sección 4.6.2). Un 60 % de los usuarios indicó que no comprendía su función.



**Figura 6.4:** Escenario significativo creado por un usuario de las pruebas realizadas durante el primer experimento. De izquierda a derecha: Escenario sin usar la extensión (prueba 1) y usando la extensión (prueba 3).

La última prueba fue completada por los usuarios en un tiempo de 20 minutos y 45 segundos de media, con una desviación estándar de 5 minutos y 1.96 segundos. Durante esta prueba, se observó que los 5 usuarios necesitaban un período de adaptación, en el que manejaban la extensión con lentitud. Tras pasar esta primera etapa, los usuarios mostraron un mejor rendimiento. Además, se observaron dos detalles importantes. El primero de ellos era que los usuarios dedicaron mucho tiempo a probar todas las acciones que podían ejecutar



**Figura 6.5:** Gráfico comparativo del tiempo destinado por cada usuario a la primera y última prueba.

utilizando la extensión. El segundo, que todos los usuarios tuvieron problemas con la gestión de colecciones en *Blender*. Los resultados de esta prueba, junto con los de la primera, se pueden observar en la Figura 6.5.

Por último, de la entrevista con los usuarios se obtuvieron los datos que se presentan a continuación. De las respuestas a la primera pregunta, se obtuvo que la extensión le pareció muy útil a los usuarios, ya que aportaba aleatoriedad a la distribución de los objetos y permitía colocar grupos grandes de los mismos. Varios de los usuarios mencionaron que les reportaba una gran comodidad al crear escenarios. Por ese motivo, todos ellos indicaron que si se diera el caso, lo utilizarían en un futuro para acelerar su flujo de trabajo. Por último, el único elemento de la interfaz que supuso un problema a la mayoría de los evaluados fue la creación y gestión de colecciones de *Blender* para cada grupo de objetos.

## Análisis de los resultados del primer experimento

Los resultados que reportó el primer experimento llevaron a varias conclusiones. En primer lugar, no se pudo probar que los usuarios construyeran escenarios más rápido utilizando la extensión que sin usarla. Como se menciona en la sección de resultados, el tiempo medio que tardaron los evaluados en completar el escenario sin usar la herramienta fue de 14 minutos y 8 segundos. Utilizando la extensión, el tiempo medio asciende a 20 minutos y 45 segundos. Se concluyó que esto se debía a 3 factores principales: el período de adaptación de los usuarios, la experimentación con las funciones de la herramienta y la falta de una indicación explícita por parte de los evaluadores. Además, se dedujo que los datos sobre la experiencia de los usuarios con *Blender*, obtenidos mediante una encuesta, no reportaban un dato objetivo sobre el que sacar conclusiones.

En cuanto a las instrucciones de instalación, se dedujo que varios apartados podrían ser explicados con mayor claridad. Además, se concluyó que se debían añadir nuevas instrucciones para evitar errores en la instalación en caso de que el usuario haya instalando *Blender* desde plataformas alternativas a la página oficial, como *Steam* o *Microsoft Store*.

Con relación a la interfaz de usuario, se extrajo de los resultados que dos elementos de los paneles requerían cambios. Estos dos elementos son: la utilidad de pintado de vértices y el campo de nombre de la colección. Ambas secciones de la interfaz suponían un problema. La primera de ellas, se dedujo, era poco intuitiva y no dejaba claro a la persona evaluada su funcionamiento. La segunda, por otro lado, tenía un gran impacto negativo si los usuarios no cambiaban de colección al distribuir objetos de diferentes grupos por el escenario. En este último caso, se concluyó que el panel debía indicar al usuario que el nombre de la colección es un factor a tener en cuenta en todo momento.

Tras las conclusiones generales que reportaron toda esta información, se recogieron todos los datos pertinentes para la continuación del proceso de evaluación. Una vez obtenidos estos datos, se pudo diseñar el segundo experimento, corrigiendo los factores que evitaban obtener los resultados deseados.

### 6.2.2. Segundo experimento

El segundo experimento se realizó tras conocer y analizar los resultados del primero. Este experimento sigue la estructura del anterior, sin embargo, modifica los aspectos de dicho experimento que provocaban una alteración en los datos obtenidos. Asimismo, enfoca

las pruebas con la intención de medir la velocidad del usuario al crear los escenarios.

En primer lugar, por los motivos especificados en la Sección 6.2.1 en lo relativo a la experiencia del usuario con *Blender*, se modificó la encuesta que recogía el perfil de cada uno. En este experimento se cambió la medición de la experiencia del usuario. Pasó de nivel de manejo del programa a tiempo de experiencia con el mismo. De esta manera, se obtendría un dato más objetivo para analizar. El resto de elementos de la encuesta se mantendrán, de manera que se pueda continuar clasificando el perfil del usuario contando con género, edad y ocupación (visitar Sección C.2).

La siguiente secuencia seguirá los mismos pasos que la segunda prueba del primer experimento. En ella, los usuarios tendrán que replicar, sin utilizar la extensión, un escenario similar al que se presenta en la Figura 6.1. En este caso, los usuarios tendrán un tiempo de 10 segundos en los que podrán ver la imagen de referencia. Tras esos 10 segundos, se eliminará esa imagen y se aportará una imagen borrosa (Figura 6.6) del mismo entorno. El motivo tras este cambio es el de evitar que el usuario reproduzca el escenario de forma casi idéntica al mostrado inicialmente, como ocurría en el primer experimento. Esta modificación se basa en la peculiaridad del comportamiento humano que tiende a imitar lo que ve [53]. En dicho estudio, y otros posteriores [54], se observaba la necesidad humana inconsciente de imitar acciones o pasos, incluso cuando se sabe que no es necesario para lograr su objetivo. Tras entregar la nueva imagen, de nuevo se comenzará a medir el tiempo que tardan los usuarios en realizar el escenario. Aparte, en este segundo experimento, se interpretará el tiempo con respecto al número de elementos repartidos por el escenario, de forma que se obtengan datos que se puedan comparar en igualdad de condiciones con los que se extraigan de la última prueba.

Durante la fase de instalación de la herramienta se procederá de la misma manera que en el experimento anterior. En este segundo experimento, las instrucciones de instalación han sido modificadas, de forma que incluyen todos los casos de instalación que supusieron un problema durante el primer experimento. Además, se han explicado con más detalle los elementos que se han considerado poco intuitivos.

La siguiente prueba procederá de igual manera que en el anterior experimento. Se facilitará a los usuarios evaluados el mismo memo con instrucciones básicas de uso (Figura 6.2). Tras esta prueba, se hará una explicación detallada y ejemplificada de todas las funciones de la extensión. Una vez se haya explicado todo, se dejará al usuario experimentar con la



**Figura 6.6:** Escenario de ejemplo borroso utilizado en las pruebas.

extensión. En este segundo experimento se comprobará que el usuario se acostumbra al manejo de la herramienta y, cuando el evaluador considere que el control del evaluado es suficiente (no duda entre acciones y conoce el flujo de trabajo), se procederá a la última prueba.

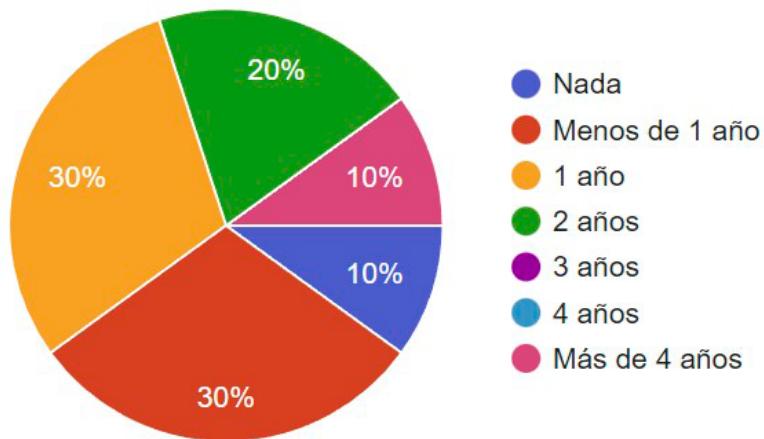
En la última prueba, el usuario tendrá que replicar el mismo escenario que se utilizó para el primer experimento. En este caso, se cuenta con un usuario que conoce las funciones de la herramienta y está habituado a ellas. En esta ocasión además, se indicará al usuario que debe realizar la prueba en el menor tiempo posible. El tiempo será cronometrado para su posterior análisis. De la misma manera que en la primera prueba, se tendrá en cuenta el número de elementos que han sido distribuidos.

Finalmente, se realizará una pequeña entrevista a los usuarios con las mismas preguntas que se realizaron durante la primera etapa de experimentación. En este caso se ha añadido una pregunta más, de manera que se pueda conocer si el usuario considera, como se espera,

que el resultado obtenido mediante el uso de la extensión es mejor que el resultado hecho a mano. La pregunta es la siguiente:

- “¿Cuál de los resultados prefieres, valorando la eficiencia, precisión y flexibilidad?”

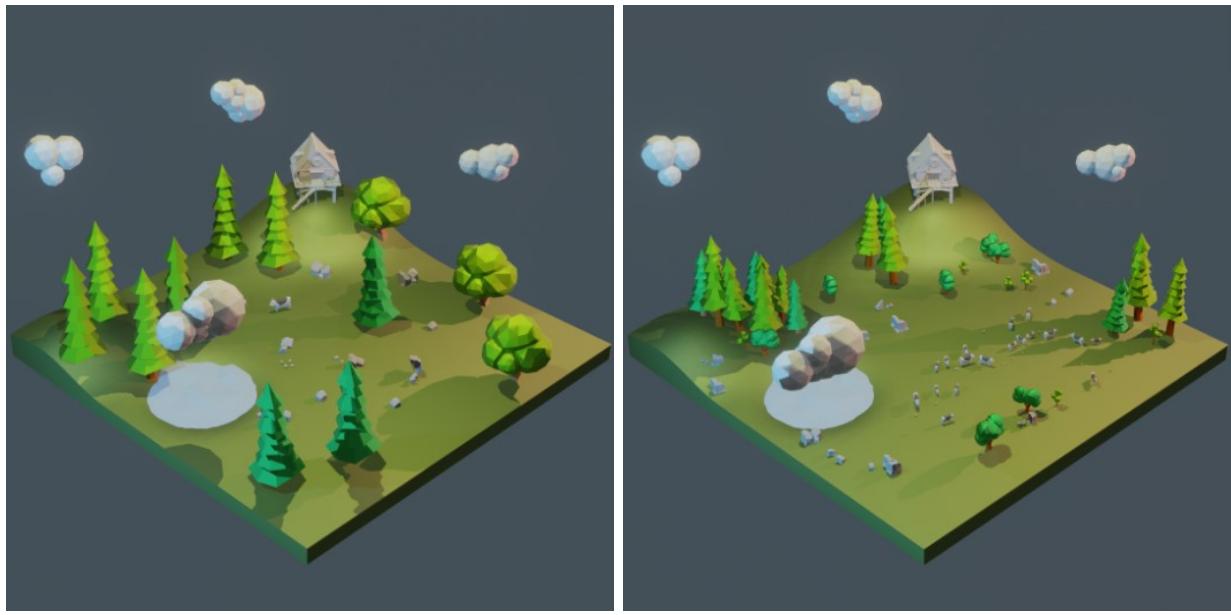
El segundo experimento fue llevado a cabo con 10 usuarios, residentes en ese momento en Madrid, España, siendo un 70 % hombres, un 10 % mujeres y un 20 % que no se identificaban con ninguno de los anteriores. Los usuarios eran distintos a los que participaron en el primer experimento y tenían un rango de edad entre 21 y 25 años, de los cuales 8 eran estudiantes del *Grado en Desarrollo de Videojuegos* en la *Universidad Complutense de Madrid* y 2 de ellos eran diseñadores de profesión. Un 10 % de los usuarios no tenía nada de experiencia de uso del programa *Blender*, un 30 % tenía menos 1 año de experiencia y otro 30 % tenía 1 año de experiencia. El resto de usuarios, que conforman la minoría, tenían más de 2 años de experiencia. El desglose de porcentajes se puede comprobar en la Figura 6.7. Por otro lado, un 60 % de los usuarios evaluados se identificaba con un perfil artístico, mientras que el 40 % restante se alineaba con un perfil orientado a la programación.



**Figura 6.7:** Gráfico representativo de la experiencia de los usuarios del segundo experimento.

### Resultados del segundo experimento

Tras seguir las pruebas que conforman el segundo experimento, se obtuvieron una serie de resultados. Los 10 usuarios tardaron de media de 9 minutos y 34 segundos, con una desviación estándar de 5 minutos y 13.21 segundos, en completar la primera prueba. Al igual que en el experimento anterior, los datos de cada usuario se pueden comprobar en una gráfica (véase Figura 6.9). En este experimento, los usuarios crearon escenarios similares al

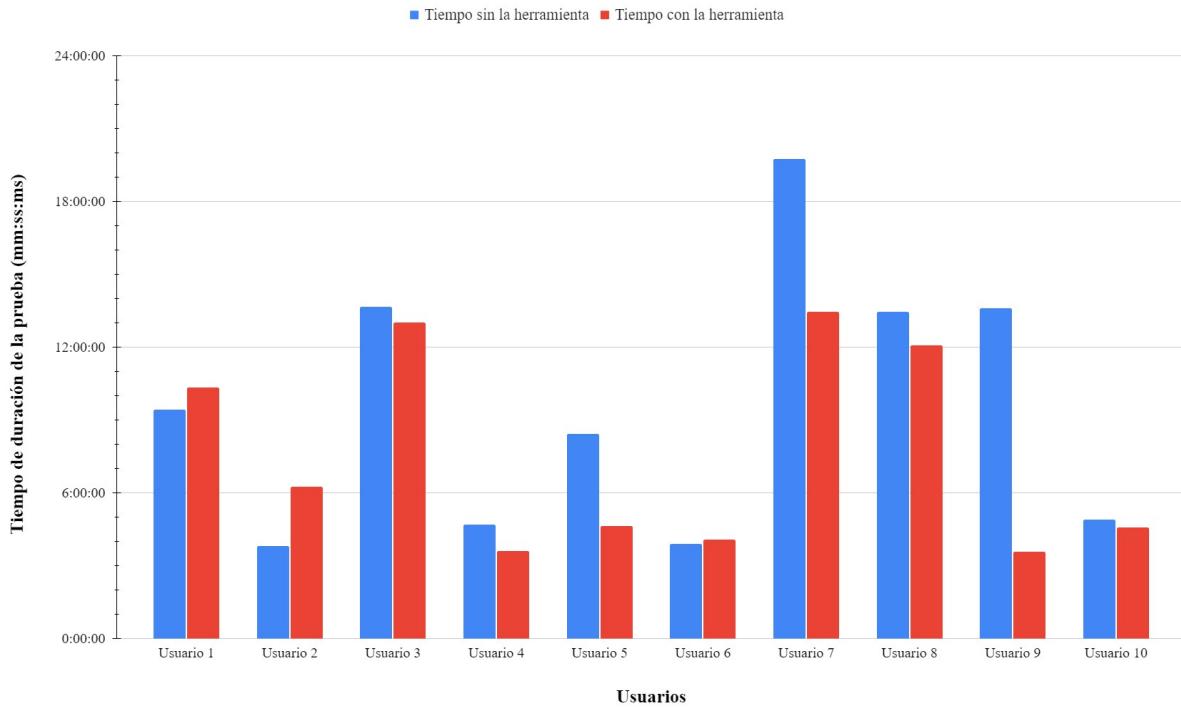


**Figura 6.8:** Escenario significativo creado por un usuario de las pruebas realizadas durante el segundo experimento. Escenario sin usar la extensión (prueba 1) a la izquierda y usando la extensión (prueba 3) a la derecha.

que se presenta en la Figura 6.8, pero, en esta ocasión, eran más variados que los obtenidos durante el primer experimento. La media de elementos distribuidos en esta prueba por los usuarios es de 28,6 elementos, con una desviación estándar de 12,22. Con relación a la instalación, todos los usuarios pudieron llevarla a cabo sin problema.

Al igual que durante el primer experimento, durante la siguiente fase de evaluación los usuarios no exploraron en un inicio los elementos que no se les habían explicado, con excepción de 30 % de los usuarios, que trata de tocar las reglas de los objetos. Durante esta parte, un 20 % de los usuarios muestran confusión respecto al control de la utilidad que simplifica el pintado del peso de los vértices (véase Sección 5.2). Tras aportar una exposición más detallada de las funciones de cada panel, a un 20 % de los usuarios no le quedó claro que las reglas afectan individualmente a cada elemento. Además, un 30 % de ellos echaba en falta la posibilidad de modificar las reglas de varios objetos al mismo tiempo.

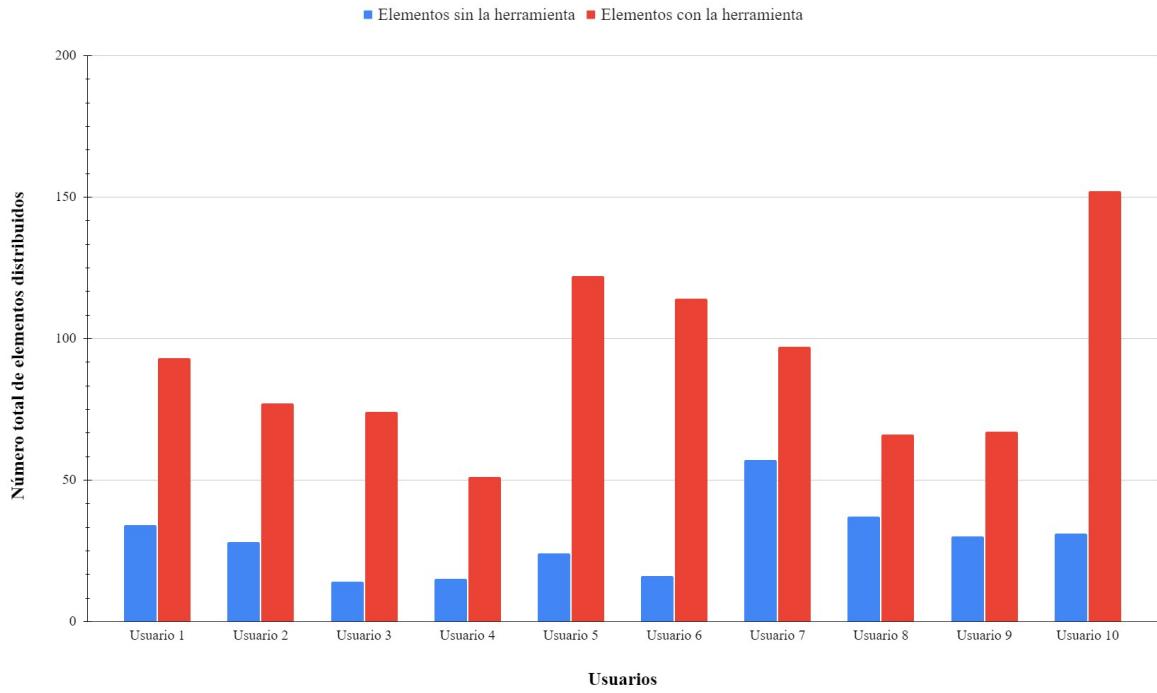
La última prueba fue completada por los usuarios en un tiempo de 7 minutos y 34 segundos de media, con una desviación estándar de 4 minutos y 8.57 segundos. Durante esta prueba, se observó que los usuarios no tenían ningún problema en alcanzar el resultado que esperaban tras la fase de evaluación previa. Todos los usuarios se adecuaron correctamente a



**Figura 6.9:** Gráfico comparativo del tiempo destinado por cada usuario a la primera y última prueba.

los objetivos establecidos, completando la prueba sin problema. Además, se midió la media de objetos distribuidos por los usuarios, siendo ésta de 91,3 elementos, con una desviación estándar de 29,2. Estos datos se pueden ver gráficamente en la Figura 6.10.

De la entrevista final se obtuvieron resultados similares a los que se consiguieron con el primer experimento. De la primera pregunta se obtuvo una respuesta común entre todos los evaluados, que indicaba que les había resultado más cómodo, más rápido y más práctico el proceso de creación del escenario haciendo uso de la herramienta. De la segunda pregunta se obtuvieron resultados variados. El 70 % de los usuarios indicó que utilizaría la extensión para crear escenarios rápidos de carácter natural, como pueden ser bosques y cuevas. El 20 % de los usuarios indicaron que les sería útil para agilizar la creación de entornos estáticos en el ambiente profesional. Todos los usuarios indicaron, como respuesta a la tercera pregunta, que comprendían todos los aspectos de la interfaz, especialmente tras la explicación detallada de la extensión. Por último, en relación a la pregunta adicional de este experimento, todos los usuarios confirmaron que el resultado generado por la extensión era mejor en términos de eficiencia, y flexibilidad. En cuanto a la calidad, establecieron que el resultado de la

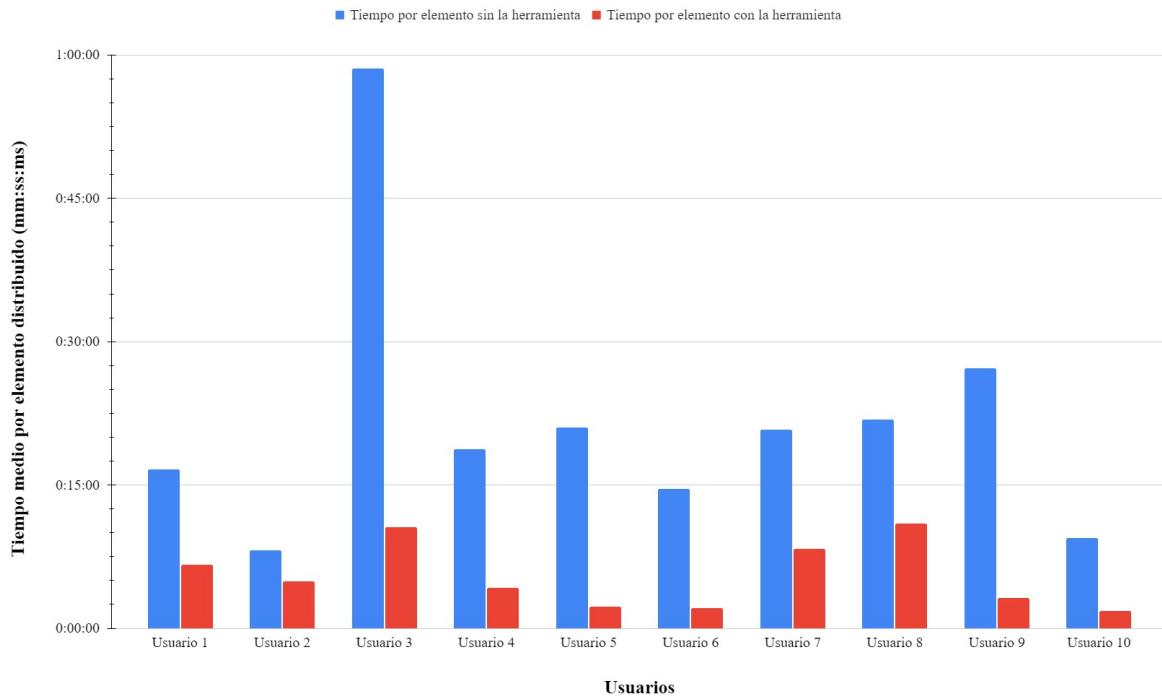


**Figura 6.10:** Gráfico comparativo de los elementos distribuidos por cada usuario en la primera y última prueba.

extensión reportaba una satisfacción igual, como mínimo, al obtenido sin la extensión.

### Análisis de los resultados del segundo experimento

Los resultados que reportó el segundo experimento llevaron a varias conclusiones. En primer lugar, se procesaron los datos obtenidos sobre el número de objetos distribuidos y tiempo tardado, tanto en la primera prueba como en la segunda. Se han comparado las cifras obtenidas de ambos factores para determinar el tiempo que ha dedicado cada usuario a distribuir un objeto de media. De esta manera, se obtiene una serie de datos comparables entre la primera prueba y la última. Los datos recogidos tienen como objetivo demostrar el incremento de la velocidad del usuario. Esto se ve representado gráficamente en la Figura 6.11. En la primera prueba se ha determinado que la media de tiempo por elemento distribuido es de 21.58 segundos por objeto, con una desviación estándar de 13.49. Por otro lado, en la última prueba, la media resultante de los datos es de 5,5 segundos, con una desviación estándar de 3,29. Si bien de los datos sobre tiempo total de las pruebas recogidos ya se podía determinar que el uso de la extensión conllevaba un 24,26 % de mejora



**Figura 6.11:** Gráfico comparativo de la media de tiempo por elemento distribuido por cada usuario en la primera y última prueba.

en tiempo respecto a la realización de un escenario sin su uso, con estos resultados se puede observar que, teniendo en cuenta el número de objetos distribuidos, la mejora es de un 74,48 %. Además, todos los usuarios han presentado un tiempo con una variación mínima, independientemente de la experiencia de cada uno con el programa *Blender*.

Los resultados de la instalación indican que los problemas que presentaban las instrucciones durante el primer experimento han sido corregidos correctamente. Se ha considerado que no se requiere ninguna indicación adicional a las instrucciones actuales. Asimismo, tras desarrollar un uso más cómodo del apartado de colecciones en la interfaz gráfica de usuario (véase Sección 5.3), ningún usuario ha mencionado que este aspecto ralentizase su flujo de trabajo, al contrario que cuando se abordó el mismo tema durante el transcurso del primer experimento.

De las respuestas a la entrevista se destaca que los usuarios son plenamente conscientes de la mejora que supone la utilización de la herramienta. Lo más significativo de las respuestas es que todos los usuarios están de acuerdo en que el resultado obtenido es, como

mínimo, igual de satisfactorio con el uso de la herramienta que sin ella. Esto cumple uno de los objetivos de partida del proyecto que, junto a la mejora en velocidad, permite obtener una resolución positiva.

En conclusión, los resultados de estas pruebas alcanzan los objetivos que se pretendían completar durante el transcurso del periodo de evaluación. A partir de estos resultados se ha determinado si las hipótesis planteadas originalmente se cumplen. Esto se desarrolla en más detalle en el Capítulo 8.

# Capítulo 7

## Discusión

En este capítulo se expondrán todos aquellos apartados del trabajo que han sido llevados a cabo como se pretendía en una primera instancia. De igual forma, se hablará de todos aquellos aspectos que no cumplen las expectativas originales o deben ser pulidos. Para ello, se comenzará con los aspectos metodológicos, referentes a la tecnología utilizada, que han permitido el desarrollo de la extensión.

En primer lugar, la utilización del programa de modelado *Blender* reportó una serie de ventajas que permitieron que el desarrollo del trabajo cumplierse ciertos objetivos. Principalmente, la extensa documentación del programa y su comunidad facilitó conocer en profundidad el funcionamiento interno del mismo y la resolución de errores. Por otro lado, contiene una *API* (definida en 1.4) capaz de realizar tareas sofisticadas relacionadas con la manipulación de mallas y elementos visuales. Esta última permite crear componentes en la propia interfaz del programa, simplificando el diseño de la entrada de usuario, descrita en la Sección 5.3. Sin embargo, *Blender* cuenta con ciertas restricciones que dificultan el correcto desarrollo de aplicaciones aditivas o herramientas. En el caso de esta extensión, la limitación con mayor impacto en cuanto a rendimiento es que *Blender* no soporta procesamiento múltiple, también conocido en notación inglesa como *multithreading*. Este factor supone que el rendimiento de la extensión, a pesar de que ya es adecuado, pueda verse mejorado en gran medida. Para abordar esta limitación, se ha diseñado la búsqueda de la solución de manera que evite explorar excesivamente todos los estados posibles (véase Sección 4.4).

También se debe destacar que los algoritmos orientados a la distribución, si bien todos

ellos cuentan con ventajas, también poseen inconvenientes. Tanto unos como los otros se desarrollan en mayor profundidad en la Sección 4.3. Originalmente, se planteó el uso de estos algoritmos de forma que permitiesen, por un lado, una distribución rápida, y por el otro, soluciones variadas, tal y como se mencionó en la misma sección. En todas las ocasiones, cada algoritmo obtiene una solución distinta. Dada la subjetividad que presenta la preferencia de los usuarios respecto a esto, se ha decidido dar al usuario la opción de seleccionar el algoritmo que quiere usar para cada escenario. De esta manera, el usuario puede realizar un proceso iterativo en el que pruebe la distribución con cada algoritmo, seleccionando el resultado que se aproxime al resultado que espera.

Con relación a la implementación de reglas de distribución, se alcanzó el resultado deseado en las que son propias de cada objeto (véase Sección 4.6.2). En el caso del resto de reglas, se encontraron una serie de limitaciones que disminuyen la precisión de la distribución. La regla sobre la superposición de objetos a distribuir, concretamente, presenta una restricción. Si para comprobar que se cumple, se utilizasen mallas complejas que delimitan el objeto, el coste computacional afectaría gravemente al rendimiento de la extensión. En su defecto, se utilizan volúmenes delimitadores simples que conllevan una distribución menos precisa, pero poseen un coste computacional que no afecta en gran medida al rendimiento de la extensión. Sin embargo, se ha decidido permitir que el usuario tome la decisión sobre si se usará una caja delimitadora o una esfera delimitadora debido a los motivos detallados en la Sección 4.6.3.

Cabe mencionar que la instalación de la herramienta requiere la utilización de un paquete de código adicional que se encarga de instalar la biblioteca *AIMA* (véase Sección 5.4), así como la extensión en sí. La necesidad de este elemento adicional proviene del extenso proceso que originalmente el usuario debía completar para comenzar el uso de la herramienta. Si bien se podía seguir sin problema, se ponderó que suponía un obstáculo considerable para el usuario.

Un segmento de la extensión parte de la idea de que todo lo que se ha creado en código pueda ser extrapolado a diferentes programas de modelado 3D. A lo largo del desarrollo, se ha dilucidado que los elementos del proyecto pertenecientes a la interfaz de usuario deben ser recreadas en las distintas aplicaciones destino en las que se pretenda implementar de nuevo la herramienta, procurando enlazar cada elemento con su función correspondiente. Este también es el caso de una serie de funciones específicas al programa en cuestión, que requerirán de adaptación al funcionamiento del nuevo entorno. No obstante, superados estos

primeros requisitos para la implementación, el resto de la herramienta se comportará de manera prácticamente idéntica dentro del programa extrapolado. Esto se debe a que la lógica interna de la distribución, la relacionada con el uso de los algoritmos y *AIMA*, permanecerá intacta entre versiones del programa.

En cuanto a la fase de evaluación y experimentación, una clara limitación es el bajo número de usuarios que se evaluaron. La edad y ocupación de todos ellos compartían mucha similitud, ya que cursaban estudios similares. Aunque el segundo experimento tratase de suplir los problemas del primero, tanto en cantidad de usuarios como en variedad de los mismos, se considera que se podrían obtener resultados mucho más concluyentes de grupos todavía más grandes y con una distinción más clara en cuanto a su profesión o estudios. Para el estudio realizado, sin embargo, se ha considerado que la experimentación realizada permite obtener conclusiones fundadas sobre las hipótesis planteadas. Esto se debe a que, aunque el rango de edades y ocupaciones fuera similar, todos los usuarios forman parte de la categoría de público objetivo del trabajo.

Otra limitación que se tuvo en cuenta durante el periodo de evaluación, como se comenta en el capítulo correspondiente (ver Sección 4.6.3), es que no se obtienen datos en los experimentos propuestos sobre los diferentes algoritmos. Aunque se trata de una parte importante del proyecto, se decidió simplificar las pruebas extrayendo de las mismas la elección de los usuarios sobre los algoritmos utilizados. Esto se debe a que el foco de estos experimentos se centraba en obtener resultados sobre la satisfacción del usuario y su flujo de trabajo en base, principalmente, al tiempo que tardaban en realizar un escenario. Aun así, como se considera que un experimento centrado en determinar cuál de los algoritmos satisface más a los usuarios reportaría datos sustanciales sobre el valor de la extensión, se ha planteado en la Sección 8.2, que recoge el trabajo futuro.

# Capítulo 8

## Conclusiones y trabajo futuro

Este capítulo tiene como propósito exponer las conclusiones derivadas de la elaboración de este trabajo, con el objetivo de alcanzar conclusiones en base a la información descrita hasta el momento. Además, se establecerán pautas para posibles trabajos futuros que busquen utilizar y potencialmente continuar con el desarrollo de este proyecto.

### 8.1. Conclusiones del proyecto

El objetivo principal de este trabajo gira en torno a la elaboración de una herramienta de distribución de objetos sobre superficies dentro del programa *Blender*, pero con la capacidad de ser extrapolable a múltiples plataformas similares. Para alcanzar dicho objetivo se han estudiado y aplicado diferentes conceptos y algoritmos de búsqueda de caminos relacionados con la inteligencia artificial. Además, se ha hecho uso del programa de modelado y animación pertinente para el desarrollo. De esta forma, se han completado todos los objetivos iniciales orientados al diseño e implementación.

Con el propósito de analizar el correcto funcionamiento de la herramienta, se han realizado una serie de pruebas con usuarios ajenos al proyecto, iterando sobre los resultados. El primer experimento fue realizado para conocer los criterios que debían seguirse durante el segundo experimento, de forma que este último reportaba resultados más concluyentes.

Una vez se ha creado la herramienta, se han alcanzado todos los objetivos y se han realizado los experimentos pertinentes, se ha estudiado si su uso cumple las hipótesis planteadas

en la Sección 1.2. Debido a que no se ha contado con una gran variedad de usuarios evaluados durante la fase de pruebas, se ha determinado que sólo se pueden validar las hipótesis parcialmente.

La primera finalidad de la fase de experimentación era comprobar si la hipótesis H<sub>1</sub> era validada. Esta hipótesis define que, utilizando la herramienta de distribución de objetos sobre superficies desarrollada, un usuario es capaz de crear escenarios tridimensionales de manera más rápida a si lo hiciese sin utilizar la herramienta. En términos de tiempo por objeto distribuido en una escena, la utilización de la extensión ha supuesto, según los datos recogidos, un 74,48 % de mejora respecto a si no se usase la extensión desarrollada. Si bien estos datos pertenecen exclusivamente a un experimento, tras el posterior análisis de los resultados obtenidos, se puede confirmar que esta hipótesis queda parcialmente validada.

La segunda finalidad de esta fase fue confirmar si se cumple la hipótesis H<sub>2</sub>. Esta hipótesis expresa que la mayoría de los usuarios encuentran que la calidad de los resultados obtenidos por la herramienta es igual o superior, en comparación a la calidad de los resultados que se pueden conseguir a mano, valorando la eficacia, precisión y flexibilidad del proceso. El análisis final de los resultados del proceso de evaluación confirma que esta segunda hipótesis se valida parcialmente. Todos los usuarios que participaron en el segundo experimento confirmaron que el resultado que obtuvieron utilizando la extensión resultó, al menos, igual de satisfactorio que el que obtuvieron sin utilizarla.

En resumen, el trabajo cumple los objetivos establecidos. Como resultado del proceso de evaluación, se demuestra que los datos obtenidos son positivos y, si bien las hipótesis solo se pueden validar parcialmente, los resultados son favorables.

## 8.2. Trabajo Futuro

El trabajo a futuro de este proyecto se presenta abierto a mejoras y ampliaciones de sus funcionalidades. Una de las principales metas es la de permitir que gran parte de las reglas de distribución puedan aplicarse incluso después de haber realizado la distribución en sí. Esto brindaría a los usuarios una mayor flexibilidad y control sobre la disposición de los elementos en sus escenas. Actualmente, unas pocas reglas permiten esta funcionalidad, como por ejemplo, el control de la adherencia de los objetos a las normales de la malla (véase Sección 4.6.2). Además, se podría añadir una regla que se encargue de limitar la distancia máxima entre objetos que siempre tenga en cuenta que la solución acabe siendo satisfactoria.

Un aspecto importante a abordar es la correcta transformación de las volúmenes delimitadores en las mallas de los objetos, concretamente aquellas con topología compleja. De esta manera, se garantizaría una representación más precisa de los objetos en el entorno, evitando distorsiones o problemas visuales que puedan surgir, como la colisión con otros objetos de la escena. Esto se podría lograr mediante la implementación de un sistema automatizado que compute diferentes volúmenes delimitadores para las mallas. Así, los usuarios podrían seleccionar la opción más adecuada a sus necesidades y al entorno en el que se realiza la distribución. Adicionalmente, se podría añadir un sistema mecanizado que determinase qué volumen sería más adecuado para cada objeto. Como objetivo más cercano y con un coste en tiempo mucho más bajo, se podría comenzar permitiendo que las cajas delimitadoras puedan ser utilizadas en todos los contextos de cada objeto, teniendo en cuenta tanto escalas, como rotaciones, como interacción con el resto de reglas.

Entre otras de las posibles mejoras, se encontraría el facilitar la incorporación de nuevos algoritmos por parte de los usuarios. Para ello se proporcionarían herramientas y documentación clara que permitiera a los posibles usuarios el contribuir con sus propias implementaciones, enriqueciendo así la variedad de opciones disponibles para sí mismos o la comunidad.

Por otro lado, actualmente la herramienta presenta, bajo situaciones extremas, un bajo rendimiento. Por ende, se podría buscar mejorar el rendimiento y la eficiencia de la herramienta mediante la adaptación del cómputo de los algoritmos a un entorno que permita procesamiento simultáneo como C++. Dado que *Blender* no soporta la implementación con múltiples hebras de ejecución (véase Sección 7), sería necesario extrapolarlo para permitir la incorporación de hilos. Esto permitiría aprovechar al máximo los recursos del sistema y acelerar los procesos de distribución y cálculos asociados.

Aparte de todo esto, habría que valorar la incorporación de elementos que mejoren el mantenimiento y la calidad del proyecto. Esto incluye mejoras en la interfaz de usuario, accesibilidad y funcionalidades adicionales que hagan que el uso del programa sea más intuitivo, eficiente y agradable. Este es un apartado importante, puesto que estas mejoras surgirían de las propias necesidades de los usuarios que hayan probado la herramienta y que no se han contemplado durante el desarrollo de la misma. Además, a esto se suma la corrección de errores, que sería una tarea constante durante el desarrollo del proyecto. Se prestaría especial atención en resolver cualquier problema o mal funcionamiento que pueda surgir durante su uso, para garantizar una experiencia fluida y libre de inconvenientes.

La interfaz, por otro lado, podría ser adaptada para otros programas de modelado 3D. De la misma manera, se podrían extraer las utilidades de la extensión que se explican en la Sección 5.2. Esta adición se plantearía exclusivamente para aquellos programas que no posean una herramienta de distribución bien desarrollada.

También se cuenta con una sugerencia propuesta por varios usuarios que utilizaron la herramienta. Se propuso que la misma selección de los vértices, mediante el pintado de pesos (véase Sección 4.1), indique una probabilidad de si un objeto podrá ser posicionado o no en un vértice concreto. Por ejemplo, si un vértice tiene un peso de 0.3, cualquier objeto tendrá una probabilidad del 30 % de ser posicionado en esa localización. Además, se planteó que pudiera en un futuro haber más de un tipo de selección de vértices. Para este trabajo sólo se optó por ser una selección aleatoria. Por este motivo, en la Figura 5.6 se presenta *Vertex Selection* como un desplegable. Esta sugerencia busca brindar a los usuarios una mayor flexibilidad y control sobre la distribución de los objetos, permitiéndoles ajustar las probabilidades de colocación en función de sus preferencias y necesidades específicas.

En cuanto a obtener mejores resultados del estudio, se deberían formular nuevos experimentos. Éstos serían planteados de forma que se puedan suplir las limitaciones encontradas en los que ya se han llevado a cabo. En primer lugar, se propondría un experimento centrado en analizar el rendimiento y grado de satisfacción de los usuarios utilizando diferentes algoritmos. Tras los resultados, se podrían sacar análisis referentes a los resultados de dichos algoritmos. Por otro lado, se podría llevar a cabo un experimento sobre el proceso de aprendizaje del usuario. Si bien se ha medido el desempeño del usuario una vez conocía la extensión, no se ha puesto un enfoque en estudiar la adaptación del mismo desde su primera interacción. Así pues, sería posible monitorizar el periodo de adecuación del usuario, cuyas conclusiones permitirían conocer en mayor medida los elementos del programa que requieren de una mejor explicación.

Por último, sería óptimo crear un experimento que se enfoque en la utilidad de la herramienta teniendo en cuenta escenarios de mayor o menor tamaño y de entornos naturales o artificiales. Para este experimento, se establecería un límite, tanto inferior como superior, de elementos que un usuario podría distribuir. De este experimento se podrían extraer conclusiones sobre qué tamaño de escenarios se benefician en mayor medida del uso de la extensión, así como qué tipo de objetos se ven favorecidos por la aleatoriedad de la misma.

En resumen, todas estas mejoras garantizarían un programa más potente, versátil e

intuitivo para los usuarios, facilitando así la realización de tareas tediosas y permitiendo prestar más atención al proceso creativo y artístico de su trabajo. En cuanto al estudio que se hace sobre el programa, se podrían plantear varios experimentos adicionales que se formulen teniendo en cuenta una fase de evaluación con mayor enfoque en los diferentes apartados. Aunque estos apartados se aparten del estudio original, se considera que aportarían valor al resultado final.

# Capítulo 9

## Contribución de los miembros

A continuación se detallarán las aportaciones de cada uno de los miembros del proyecto, destacando qué tareas fueron llevadas a cabo en grupo y cuáles individualmente. Las tareas fueron divididas en función de las aptitudes y preferencias de cada uno de los integrantes del equipo.

### 9.1. Jose Daniel Rave Robayo

Jose Daniel estuvo presente en las reuniones bisemanales con el resto de integrantes del grupo y los tutores del mismo. De la misma manera, participó en la posterior asignación de tareas siguiendo una organización y priorizando las tareas cruciales del proyecto. Cabe mencionar que Jose ideó el concepto del proyecto en un principio, el cual con ayuda de los integrantes y tutores, fue siendo modificado conforme avanzaba el mismo.

Por supuesto, junto con los miembros del equipo, se encargó de barajar qué tecnologías utilizar para el desarrollo de la herramienta. Por ejemplo, introdujo la extensión de *VS Code* (ver Sección 1.4.1) que permitió agilizar el desarrollo. De la misma manera, al ser propuesto usar la biblioteca *AIMA* (ver Sección 2.2.2), se encargó de introducirla dentro del proyecto, de manera que *Blender* reconociese el nuevo paquete de código.

Una vez iniciado el proyecto, Jose Daniel implementó, junto con Sergio y Daniel Illanes, el primer prototipo de distribución, el cual se encuentra descrito en la Sección 4.1. Este primer prototipo implicó adentrarse y conocer el funcionamiento de *Blender*, pues hacia

falta extraer toda la información necesaria del entorno 3D. Además, esto permitió a Jose Daniel conocer las distintas funcionalidades que ofrece la *API* de *Blender*, como crear copias por referencia, asignar pesos a los vértices mediante el *Weight Paint*, o almacenar objetos en distintas carpetas (conocidas en *Blender* como colecciones). Si bien es cierto que no son utilidades vitales, facilitan ciertas tareas y mantienen un orden dentro del entorno, que a largo plazo puede tener un gran impacto.

Más tarde, cuando ya hubo una primera distribución funcional, junto con los demás integrantes, se hizo uso de las elementos integrados en *AIMA* (consultar Sección 4.2). En concreto, Jose Daniel se encargó de explorar el funcionamiento de esta nueva biblioteca, redefiniendo algunos de los métodos necesarios que demandaban los algoritmos utilizados (ver Sección 4.3). El proceso conllevó, por un lado, entender cómo las posibles acciones (producidas por el método *Acciones* de cada estado) influían considerablemente en el tiempo de búsqueda de una solución, y por el otro, un estudio exhaustivo de qué algoritmos eran los más aptos al problema en función de su naturaleza. Esto supuso idear y diseñar el comportamiento del problema en cuestión, de manera que fuese capaz de producir el número de soluciones solicitadas por el usuario (ver Sección 4.4) mediante los métodos básicos de *Problem*. De la misma manera, la estrategia creada, cuya representación se puede ver en la Figura 4.4, permitió que las soluciones encontradas fuesen diferentes entre ellas. Posteriormente Daniel Illanes se hizo cargo de hacer posible seleccionar las soluciones encontradas por los algoritmos.

En cuanto a las acciones y métodos, estos últimos fueron estructurados con ayuda de Sergio, el cual se encargó de implementar las restricciones y reglas que siguen tanto los objetos como la propia distribución. En sus contribuciones se concretan otros aspectos de este desarrollo. Jose, mientras tanto, añadió los datos necesarios que definen en sí un estado en una distribución, de manera que fuese compatible con las acciones que potencialmente iban a aplicarse.

Finalizando con la distribución, Jose Daniel se encargó de implementar varias funcionalidades dentro del programa 3D que otorgarían a la extensión de flexibilidad y personalización. Las funcionalidades eran principalmente gráficas, de entre las cuales las más importantes son (1) permitir la creación de múltiples perfiles de pintado por peso (campo *Profile* de la Figura 5.4), (2) listado de objetos que parcialmente forman parte de la solución (Figura 5.7) y selección del algoritmo a utilizar. Cabe resaltar que quién se encargó mayoritariamente de la interfaz de usuario fue Sergio.

La extensión, al usar el paquete de código externo *AIMA*, precisaba de un proceso de incorporación distinto al de otras herramientas. Por lo que Jose decidió realizar una serie de pasos de instalación con el objetivo de que cualquiera usuario fuese capaz de instalar la herramienta. Sin embargo, gracias al consejo de sus compañeros, estas normas no eran lo suficientemente accesibles para todo el público, ya que implicaba ciertos conocimientos y manejos técnicos. Por consiguiente, Jose se encargó de implementar una extensión auxiliar muy breve que se encargase de instalar la extensión principal. Toda la información al respecto se encuentra en el Apéndice A.

Para finalizar, los tres integrantes del proyecto se ocuparon de llevar a cabo las pruebas con usuarios y sus posterior análisis. Aunque hay que dar el mérito a Sergio por asumir el cargo de realizar las gráficas y extraer las estadísticas necesarias de dichas pruebas. Entre otras tareas comunes se encuentran la redacción y estructura de la memoria, así como la revisión de la misma. Cada uno de los miembros se encargó de explicar y redactar los capítulos relacionados con sus tareas a lo largo del proyecto.

## 9.2. Sergio José Alfonso Rojas

Sergio ha participado durante todo el transcurso del proyecto, junto con sus compañeros, en la organización bisemanal de tareas que fueron repartidas. Además, ha tomado parte en las discusiones sobre las decisiones de diseño que se deberían llevar.

Al inicio del proyecto, los tres integrantes del equipo se encargaron de determinar qué programas y tecnologías se usarían en el proyecto. Una vez determinado, se encargaron de idear e implementar el primer prototipo de la extensión, así como estudiar cómo llevar a cabo esta tarea con la *API* de *Blender*.

Tras finalizar la fase inicial del proyecto, Sergio se encargó de implementar la primera de las utilidades: la subdivisión. Para ello, tuvo que crear dos funcionalidades. En primer lugar, programó los elementos necesarios para aplicar una subdivisión a una malla en *Blender*. De igual manera, se encargó de que la malla a la que se aplicaba la subdivisión fuera duplicada para evitar modificaciones del recurso original. En segundo lugar, estableció, mediante paneles de *Blender*, la interfaz necesaria para que el usuario pudiera acceder a esta utilidad y modificar sus atributos.

Posteriormente, Sergio se encargó, junto a Daniel Illanes, de diseñar el sistema de reglas

que deben seguir los objetos al ser distribuidos (ver Sección 4.6). Una vez se decidieron las reglas iniciales, junto con sus parámetros por defecto y sus límites, se dividió la implementación de los mismos. Sergio se encargó, en primera instancia, de crear el panel que permite la configuración de dichas reglas. Además, se encargó de desarrollar varias de estas reglas. La primera que desarrolló fue la regla que permite limitar la distancia mínima entre objetos distribuidos. A continuación, desarrolló la regla que determina qué objetos permiten que el resto de elementos se puedan superponer con ellos. Para ello, hizo uso de las cajas delimitadoras aportadas por *Blender*. Más tarde, utilizó el radio de la esfera delimitadora de cada objeto para abordar las limitaciones que presentaba la caja delimitadora, desarrollados en la Sección 4.6.

Sergio se hizo cargo de dos reglas más: el factor de escala aleatorio y el peso de aparición. Para crear el factor de escala, dividió la regla en dos apartados, coincidentes con el factor mínimo y el factor máximo entre los que se elegiría un valor aleatorio. La última regla que implementó Sergio fue el peso de aparición. Para que esta última tuviera sentido, se encargó de cambiar la estructura de las reglas, de manera que cada elemento a distribuir tuviera un conjunto de reglas propio. De esta forma cada elemento podría ser distribuido rigiéndose por restricciones diferentes.

Por otro lado, junto a Jose Daniel, Sergio se encargó de gran parte de la interfaz gráfica de usuario. Cada uno de los integrantes aportó, en este campo, los paneles referentes a las funcionalidades que habían implementado. Sin embargo, Sergio y Jose Daniel se aseguraron de que todos los paneles funcionaban correctamente y estaban diseñados de modo que los usuarios no tuvieran dificultades en su comprensión.

Por último, Sergio tomó parte en el diseño de todo el proceso de evaluación con usuarios junto con el resto de integrantes del equipo. Asimismo, participó activamente en dicho proceso, obteniendo los datos y analizándolos posteriormente. Sergio se encargó de procesar todos los datos obtenidos y comparar los resultados para obtener conclusiones finales. Además, al igual que el resto de integrantes, participó en gran medida en la escritura de la memoria del proyecto.

### 9.3. Daniel Illanes Morillas

Al igual que el resto de integrantes del grupo, Daniel participó en la repartición y planteamiento de tareas, estando presente en las reuniones bisemanales con los tutores, y

colaborando con otros integrantes cuando la tarea lo requería.

Inicialmente, Daniel se encargó de investigar y desarrollar un primer prototipo de la distribución basándose en *Poisson* (ver Sección 4.1). Esto implicó además implementar una serie de herramientas de apoyo visual en pos de facilitar la depuración. Aunque finalmente este trabajo fue descartado, parte de las bases del mismo se utilizaron para prototipar más adelante.

Posteriormente, junto a Sergio, se encargaría de diseñar el sistema de reglas que deben seguir los objetos al ser distribuidos (ver Sección 4.6). Esto implicaría implementar tanto la estructura interna necesaria para el funcionamiento de las reglas, como el apartado de interfaz de usuario que permitiera su uso. La implementación se dividió inicialmente entre los dos, resultando en el desarrollo de la regla de rotación por pasos de los objetos por parte de Daniel. Posteriormente también desarrollaría la característica de adherencia a las normales, que implicaría entre otras tareas la creación un nuevo operador que manejara dicha regla.

Más adelante se encargaría de elaborar la funcionalidad de distribución múltiple (ver Sección 5.3). Sobre este apartado se iteraría a lo largo del desarrollo del trabajo en varias ocasiones, añadiendo o adaptando nuevas características de la herramienta, o corrigiendo errores que pudieran surgir durante el desarrollo.

Asimismo, también asumiría la tarea de implementar la distribución parcial (ver Sección 5.3), adaptada para los casos de distribución única y distribución múltiple. Se apoyaría en trabajo previo de Jose Daniel para elaborar el panel referente a esta funcionalidad.

Manteniendo la relación con el apartado de distribución, y habiendo desarrollado lo anterior mencionado, también añadiría al proyecto soporte para almacenar múltiples soluciones (ver Sección 4.3.1). Este trabajo iría siendo actualizado a lo largo del proyecto a la vez que se incorporaban nuevas funcionalidades. Se apoyaría en la aportación de Jose Daniel sobre algoritmos que devuelven múltiples soluciones.

También se llevaría a cabo la tarea de permitir que los usuarios modifiquen la semilla del generador de números aleatorios. Esto se realiza a través del panel como una opción más, y de esta manera permite a los usuarios repetir generaciones “aleatorias” (ver Secciones 2.1.1 y 5.3).

Además, Daniel formó parte junto al resto de sus compañeros de realizar las pruebas de usuario pertinentes. Esto implicaba el proceso de realizar la prueba, tomar datos y analizarlos posteriormente. Por último, se encargaría de contribuir con la redacción de la memoria, así como de realizar una revisión y corrección general de la misma una vez se encontraba en vías de finalización.

# Capítulo 10

## Introduction

The design of digital environments has become a crucial task in the entertainment industry. From movies to video games, the creation of settings involves exhaustive and creative work that requires a considerable amount of time to achieve a satisfactory result.

In this regard, the main technological applications dedicated to the creation of 3D elements and their aesthetic appearance have evolved significantly over the years, incorporating increasingly sophisticated tools with the aim of enabling the construction of three-dimensional spaces. In this way, they allow for the topological generation of objects, as well as the application of textures and materials to create different styles that fit the project. However, creating a digital environment is not primarily about producing the elements that make up the scene. In addition to being an important part, the placement or distribution of these elements in the virtual space is a determining factor in achieving a good composition that defines an immersive and coherent experience for the user.

There are paid programs like *Maya* or *Houdini*, and even free-to-use programs like *Blender*, which incorporate the possibility of distributing multiple objects on a surface. However, their functionality is limited as they do not provide freedom for the user to design the environment to their liking. Additionally, it should be noted that their complexity and lack of intuitiveness can be a significant obstacle, especially for users without prior experience in using this type of software.

These programs have a wide variety of extensions that add additional functionalities and through them, provide users with a series of conveniences that they wouldn't have with

the base software. As a result, many new tools have been developed for artists and designers aiming to support the creation of both individual models and complex scenarios.

Therefore, it has been decided to develop an extension that allows the distribution of one or more objects on a given surface, relying on different parameterized algorithms manipulable by the user, in order to minimize inhibiting their creativity. For the use of multiple algorithms, an implementation of the university textbook “Artificial Intelligence: A Modern Approach” called *AIMA* will be utilized. This textbook compiles a wide variety of techniques and algorithms related to artificial intelligence, which will be employed in this document.

Furthermore, the development will be carried out in such a way that it can be extrapolated to other 3D softwares, keeping the core functionality completely separate from the program and its application programming interface (commonly known by its acronym, API).

This document will discuss the development of a tool for *Blender*. It will delve into details ranging from the idea that motivated its creation and the obstacles encountered, to the new knowledge and tools employed to carry out the project.

## 10.1. Motivation

The development of this specific work is based on the need for tools that facilitate tasks for designers and artists. This would enable them to lighten their workload and save time in the creation of settings. Additionally, it would result in increased efficiency and an improvement in the quality of their work.

Thus, implementing extensions for 3D programs has become an increasingly common and, in many cases, necessary practice, as the inclusion of new functionalities enriches the program’s content and substantially enhances it. Furthermore, this incentive is reinforced when access to the *API* is made easier, allowing the community to interact with it and collaborate in its evolution, as defined in Section 10.4.

Therefore, in order to design a solution that fully meets market expectations, it has been decided to develop a prototype using the program *Blender* (see Section 10.4.1), which allows for experimentation and the design of an optimal solution from which satisfactory results can be obtained.

The proposed solution is believed to address the aforementioned problem, as it allows artists or designers to focus more on composition and the arrangement of groups of objects, rather than individual objects, thereby enhancing production speed. Furthermore, while the tool can be useful for larger project contexts where scene creation is a vital component, it can also be utilized in projects that require more improvised and ephemeral development, such as *Game Jams*<sup>1</sup>.

In conclusion, the purpose of this work is to define an implementation style that is scalable, meaning that it allows for ongoing development as potential expansions arise. This approach ensures reaching clear goals while leaving room for users or developers to extend the functionalities.

## 10.2. Initial Hypothesis

Based on the previous section, the main objective is to create a tool that assists users in terms of time and composition when creating 3D environments. Therefore, the following hypothesis has been formulated to guide the project's direction:

**Hypothesis 1** (H<sub>1</sub>). *By utilizing the developed automatic object distribution tool on surfaces, a user is capable of creating three-dimensional scenes in less time compared to not using the tool.*

Validating H<sub>1</sub> provides information on whether the tool fulfills the stipulated goals of this project. However, this validation does not provide information on whether users are sufficiently satisfied with the result to prefer using the tool over manual methods. Therefore, the following hypothesis arises:

**Hypothesis 2** (H<sub>2</sub>). *The majority of users find that the quality of the results obtained by the tool is either better or equal compared to the quality of results achievable manually, taking into account the precision and flexibility of the process.*

---

<sup>1</sup>A *Game Jam* is an event that brings together developers with the goal of creating one or multiple videogames within a short period of time, usually between 24 to 48 hours.

### 10.3. Objectives

As previously mentioned, the work aims to develop a tool for object distribution in a three-dimensional space. With this in mind, along with the previously stated hypothesis, the following objectives are proposed:

- **Develop a first prototype that meets the minimum requirements.** A prototype program will be created that is capable of positioning objects on a given surface. Additionally, customization and flexibility will be provided to the distribution itself.
- **Establish a system of rules and restrictions that the distributed objects must follow.** Parameters related to each type of object will be defined, as well as how they can be modified to limit their distribution capacity.
- **Implement these systems using various distribution algorithms to achieve the most optimal solution and efficiency.** Iteration will be done with different algorithms, aiming to determine which one or ones generate a distribution resolution that is more useful for the artist and more versatile in terms of modifying the position of objects.
- **Implement a series of utilities for a specific program in the add-on to allow the user easier access to its main functionalities.** Tools will be provided that facilitate object placement on different parts of the surface, apply subdivisions, save distribution profiles and other possible functionalities that could be designed throughout the development.
- **Design a system that distributes objects on a surface in any 3D modeling program without distinction.** A set of classes will be established that can be accessed from different applications without exclusivity to a specific one. For this purpose, a specific modeling program will be considered to apply the solution, which theoretically could be extrapolated to other programs.
- **Implement a user interface in the specific modeling program that allows the use of this system through a hierarchical panel structure.** Different sections of the extension will be divided into panels, enabling intuitive navigation for the user.
- **Evaluate the interactions of a group of users with the add-on and make the necessary adjustments once the collected data has been analyzed.** The

profile of each user will be taken into account to determine which feedback elements will be given more consideration.

- **Draw a conclusion based on the collected data.** It will be determined whether the originally proposed hypothesis is fulfilled. Furthermore, based on this conclusion, aspects of the work that can be expanded to achieve better results will be identified.

## 10.4. Methodology

In this section, the tools and programs used for the resolution of this project will be discussed, as well as the organization of the team members to carry out the different tasks that make up the work.

### 10.4.1. Technologies and tools used

To implement the extension, first and foremost, a 3D modeling program is needed to develop the work. A 3D modeling program is a digital tool used to create three-dimensional models of objects, environments, or characters in a virtual environment (see Section 2.2). It is commonly used by professionals with an artistic profile, although technical profiles are not excluded as it also allows for the execution of physical simulations. Therefore, for this project, *Blender* has been chosen due to its versatility and user-friendly interface. Additionally, one of its standout features is that it is open-source, meaning its source code is freely available for use. This facilitates the creation of extensions and tools that enhance its functionalities. Moreover, *Blender* has an extensive and well-documented *API* (Application Programming Interface). In general terms, an *API* aims to enable communication between external elements and the internal content of a program, providing access to specific data or actions within the program.

To develop an extension focused on Blender, the interpreted language Python is used. While this may seem advantageous due to its simple syntax and visual clarity, later on, the disadvantages of this language will be highlighted, such as the lack of typing and its limited efficiency.

This was the language used for the development of the extension. While any text editor could have been used, it was decided to use the *Visual Studio Code (VS Code)* environment. VS Code is an integrated development environment (commonly known as an *IDE*), which is

a software application that implements features to facilitate the development of applications or tools and incorporates modules to work with different programming languages or even multiple platforms. One of the main reasons for choosing this *IDE* is its simplicity and elegant graphical interface. However, the main reason for not using another environment is related to the existence of an add-on for *VS Code* that facilitates the development of tools for *Blender*. This extension is called “Blender Development in *VS Code*”<sup>2</sup>. It not only provides keyboard shortcuts for creating operators (see Section 5.1), but also includes the ability to debug the code by linking the *Blender* program with the *VS Code* environment, which speeds up error detection and development.

On the other hand, as mentioned in the Introduction, this work uses the *AIMA* library, which is an implementation of the book “*AIMA: Artificial Intelligence: A Modern Approach*”[1] (see Section 2.2.2). This is undoubtedly the most important tool in the project, as it incorporates a significant portion of the functionalities and algorithms used in the add-on, enabling the distribution of objects on surfaces. Section 4.3 explains which algorithms were used, as well as other elements integrated in this library.

Lastly, for project organization, different applications were used to facilitate future task planning with the aim of carrying them out efficiently. Initially, *GitHub* was used as a version control system to maintain order and backups of the development, where the repository of this study is hosted (see Appendix Enlaces relacionados con el proyecto). Regarding task management, *Pivotal Tracker* was used to monitor their progress. Typically, the tasks had a duration of two weeks, aligning with the agreed meeting intervals with the project directors.

#### 10.4.2. Work plan

To keep track of the project’s progress and establish a working routine, it has been agreed with the directors to have regular meetings. All the planning is outlined in Table 10.1.

### 10.5. Document structure

In this section, a brief description of each subsequent chapter and section that make up the document will be provided to give an overview of the content and the order in which

---

<sup>2</sup>Add-on repository: [https://github.com/JacquesLucke/blender\\_vscode](https://github.com/JacquesLucke/blender_vscode)

Date	Work done
May 2022	Formulation of the original idea, formation of the work-group, and agreement with the project directors.
September 2022	Organization of work at regular intervals, selection of technologies to use, periodic planning sessions with the directors, and project initiation.
October 2022	Design and creation of the first distribution prototype. Initial study of the state of the art.
November 2022	Implementation of the <i>AIMA</i> library and initial algorithms. Design and implementation of the first utilities.
January 2023	Design and implementation of the first rule system and optimization of the algorithms used.
February 2023	Vertex painting utilities, advanced version of the rule system, and final configuration of the user interface.
March 2023	Final rule system, addition of the last algorithms used for distribution, and final utilities.
April 2023	Integration and coordination of all systems and start of the project memory.
May 2023	Completion of the project memory, evaluation and testing with users, and obtaining final conclusions.
June 2023	Presentation of the project to the evaluation committee and its publication.

**Tabla 10.1:** *Work plan.*

the fundamental aspects of the research will be presented.

In Chapter 2, following this section, the state of the art will be studied. It will delve into the technologies that play a role similar to what is proposed in this work and clarify which characteristics are left unexplored, demonstrating the direction in which the add-on will be developed. In addition to similar technologies and applications, the chapter will discuss the algorithms that comprise the internal structure of the work and their current status.

The architecture of the project and its organization are detailed in Chapter 3. The elements that compose the program and how they are encapsulated in types and classes are described. Additionally, the structure is argued and analyzed, along with its dependencies and limitations.

Chapter 4 will discuss the choice of distribution algorithms, their functioning, and the benefits found in the final algorithms compared to the ones initially used in the project.

Furthermore, the implementation of the *AIMA* library [1] for *Python* will be developed. On the other hand, rules and restrictions will be addressed. These are combined with the distribution algorithms explained in Section 4.3 to define how objects will be distributed on each surface.

Chapter 5 will be the final contribution chapter, discussing the workflow that should be followed when using the extension. It will explain the different panels of the user interface and how users can take advantage of all the provided utilities.

The evaluation with users will be discussed in Chapter 6, where the evaluation method will be detailed. This will include information about the evaluation process, such as the conducted experiments, the number of participating users, and their profiles. This evaluation will provide a set of results that, once collected, will lead to a section on their analysis.

Chapter 7 will develop the advantages and disadvantages of the work. It will be compared to the aspects discussed in Chapter 2. Objectives achieved and those that have not been exceeded will be determined.

As a summary, Chapter 11 will present the final conclusion reached based on the obtained results. Additionally, it will discuss the work that remains to be developed in the future.

Finally, Chapter 9 will contain the contributions made by the members of this work.

# Capítulo 11

## Conclusion and future work

The purpose of this chapter is to present the conclusions obtained from the completion of this work, in order to arrive at deductions based on the information provided so far. Furthermore, guidelines will be established for future works that aim to utilize and potentially advance the progress of this project.

### 11.1. Project Conclusions

The main purpose of this project revolves around developing an object distribution tool on surfaces using the *Blender* program, along with the ability to be used across similar platforms. To achieve this goal, various concepts and algorithms related to artificial intelligence for pathfinding have been researched and applied. Additionally, the mentioned modeling and animation software has been used for project development. As a result, all initial objectives related to design and implementation have been successfully met.

In order to assess the proper functioning of the tool, a series of tests were conducted with external users who were not involved in the project, iterating based on the results obtained. The first experiment was carried out to establish the criteria to follow during the second experiment, aiming to obtain more conclusive results.

Once the tool was created, all objectives were achieved, and the corresponding experiments were conducted to evaluate whether its use fulfilled the hypotheses stated in Section 10.2. However, due to the limited number of users evaluated during the testing phase, it was

only possible to partially validate the hypotheses.

The main objective of the experimentation phase was to verify whether hypothesis H<sub>1</sub> was validated. This hypothesis states that by using the developed object distribution tool on surfaces, a user can create three-dimensional scenarios faster than if they were to do it without using the tool. According to the collected data, in terms of time per object distributed in a scene, the use of the add-on has shown an improvement of 74,48 % compared to not using the developed add-on. Although these data are based solely on one experiment, after analyzing the results obtained, it can be confirmed that this hypothesis is partially validated.

The second purpose of this phase was to confirm whether hypothesis H<sub>2</sub> is fulfilled. This hypothesis states that the majority of users consider that the quality of the results obtained through the add-on is equal to or superior compared to manually achieved results, considering the effectiveness, accuracy, and flexibility of the process. The final analysis of the evaluation process results partially confirms this second hypothesis. All users who participated in the second experiment confirmed that the results obtained using the add-on were, at the very least, as satisfactory as those obtained without using it.

In summary, the work successfully achieves the established objectives. The results of the evaluation process demonstrate that the obtained data is positive, and although the hypotheses are only partially validated, the results are favorable.

## 11.2. Future Work

The future work of this project focuses on improving and expanding its functionalities. One of the main goals is to enable most distribution rules to be applied even after the initial distribution has been made. This would provide users with greater flexibility and control over the arrangement of elements in their scenes. Currently, only a few rules allow for this functionality, such as controlling the adherence of objects to mesh normals (visit Section 4.6.2). Additionally, a rule could be added to limit the maximum distance between objects, always ensuring that the solution remains satisfactory.

An important aspect to address is the accurate transformation of bounding volumes on object meshes, especially those with complex topology. This would ensure a more precise representation of objects in the environment, avoiding distortions or visual issues such as

collisions with other objects in the scene. To achieve this, an automated system could be implemented to calculate different bounding volumes for the meshes. This way, users could select the most suitable option according to their needs and the environment in which the distribution takes place. Additionally, a mechanized system could be added to determine which volume would be most appropriate for each object. As a more immediate goal with a lower time cost, allowing bounding boxes to be used in all object contexts could be implemented, taking into account both scales and rotations, as well as their interaction with other rules. This would facilitate the configuration and manipulation of bounding volumes, providing users with greater flexibility.

One of the possible enhancements is to facilitate the incorporation of new algorithms by users. To achieve this, tools and clear documentation would be provided, enabling users to contribute with their own implementations, thus enriching the available options for both themselves and the community.

Additionally, the current add-on may experience low performance in extreme situations. Therefore, improving the efficiency and performance of the tool could be pursued by adapting the algorithm calculations to an environment that supports concurrent processing, such as C++. Since Blender does not support multi-threaded implementation (visit Section 7), extending it to allow for thread incorporation would be necessary. This would make the most of system resources and accelerate the distribution and associated calculation processes.

In addition to everything mentioned, it would be important to consider the incorporation of elements that improve the maintenance and quality of the project. This would include enhancements to the user interface, accessibility, and additional functionalities that make the program more intuitive, efficient, and enjoyable to use. This aspect is crucial, as these improvements would arise from the needs of users who have tested the tool and were not initially considered during its development. Additionally, there would be a constant task of bug fixing to address any issues or malfunctions that may arise during its use, ensuring a smooth and seamless experience.

Regarding the interface, adapting it for other 3D modeling programs could be considered. Similarly, the utilities of the add-on explained in Section 5.2 could be extrapolated. This addition would be particularly useful for programs that do not have a well-developed distribution tool.

A suggestion has also been received from several users who have used the add-on. The

proposal is to incorporate vertex selection through weight painting (refer to Section 4.1), indicating the probability of an object being positioned at a specific vertex. For example, if a vertex has a weight of 0,3, any object would have a 30% probability of being placed at that location. Additionally, the possibility of having multiple types of vertex selection in the future has been raised. In this work, only random selection was implemented, as shown in Figure 5.6 as a dropdown option called *Vertex Selection*. This suggestion aims to provide users with greater flexibility and control over object distribution, allowing them to adjust placement probabilities based on their preferences and specific needs.

In order to get better results in the study, new experiments could be designed to address the limitations identified in the previous experiments. Firstly, an experiment could be devised to analyze the performance and user satisfaction when using different algorithms. This would allow for comparative results and analysis of the various algorithms used. Additionally, it would be optimal to conduct an experiment focused on the user learning process. While the performance of the users has been evaluated once they have acquired knowledge of the add-on, the user's adaptation from their initial interaction has not been explored in detail. Therefore, tracking the user's adaptation period would provide valuable insights into program elements that require better explanation and understanding.

Conducting these new experiments would enable the collection of more comprehensive and precise data, as well as provide additional insights into algorithm performance and user experience. This would contribute to improving and optimizing the program, as well as identifying areas that require more attention and development in future iterations.

Lastly, it would be optimal to conduct an experiment that focuses on evaluating the usefulness of the add-on in different scenarios of varying sizes and environments, whether natural or artificial. In this experiment, a minimum and maximum limit of elements that a user could distribute in the scene would be established. From this experiment, conclusions could be drawn regarding which scenario sizes benefit the most from using the add-on, as well as which types of objects are favored by the randomness provided by the tool.

In summary, all these proposed improvements would ensure a more powerful, flexible, and intuitive program for users, making tedious tasks easier and allowing them to focus more on the creative and artistic process of their work. Additionally, several additional experiments could be designed to more thoroughly evaluate different aspects of the program. Although these aspects may deviate from the original study, they are considered to add value

to the final outcome.

# Bibliografía

- [1] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.
- [2] Benjamin Bradley. Towards the procedural generation of urban building interiors. *The University of Hull*, 2005.
- [3] Joseph Alexander Brown and Marco Scirea. Procedural generation for tabletop games: User driven approaches with restrictions on computational resources. In *Proceedings of 6th International Conference in Software Engineering for Defence Applications: SEDA 2018* 6, pages 44–54. Springer, 2020.
- [4] Gillian Smith. An analog history of procedural content generation. In *FDG*. Boston, MA, 2015.
- [5] Tom Hatfield. Rise of the roguelikes: A genre evolves. *GameSpy*. Jan, 29, 2013.
- [6] Richard Moss. 7 uses of procedural generation that all developers should study. gama-sutra. *Retrieved January*, 1, 2016.
- [7] Julian Togelius, Alex J Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss, and Kenneth O Stanley. Procedural content generation: Goals, challenges and actionable steps. In *Procedural content generation*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
- [8] Ian Parberry. Designer worlds: Procedural generation of infinite terrain from real-world elevation data. *Journal of Computer Graphics Techniques*, 3(1), 2014.
- [9] Jonas Freiknecht and Wolfgang Effelsberg. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1(4):27, 2017.
- [10] David S Ebert, F Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.
- [11] Kirsten Moana Thompson. Scale, spectacle and movement: Massive software and digital

- special effects in the lord of the rings. In *From Hobbits to Hollywood*, pages 283–299. Brill, 2006.
- [12] Miles Merrill. Where's massive?: Past, present and future for the lord of the rings' crowd software. *Metro Magazine: Media & Education Magazine*, 1(139):142–145, 2004.
- [13] Alba Amato. Procedural content generation in the game industry. *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*, pages 15–25, 2017.
- [14] James E Gentle. *Random number generation and Monte Carlo methods*, volume 381. Springer, 2003.
- [15] Pierre L'Ecuyer. Random numbers for simulation. *Communications of the ACM*, 33(10):85–97, 1990.
- [16] Pierre L'ecuyer. Pseudorandom number generators. *Encyclopedia of Quantitative Finance*, 10:9780470061602, 2010.
- [17] Adi Botea, Bruno Bouzy, Michael Buro, Christian Bauckhage, and Dana Nau. Pathfinding in games. In *Pathfinding in games*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
- [18] Kairanbay Magzhan and Hajar Mat Jani. A review and evaluations of shortest path algorithms. *International journal of scientific & technology research*, 2(6):99–104, 2013.
- [19] Claude Berge. *The theory of graphs*. Courier Corporation, 2001.
- [20] Bryan Stout. Smart moves: Intelligent pathfinding. *Game developer magazine*, 10:28–35, 1996.
- [21] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015:7–7, 2015.
- [22] Xiao Cui and Hao Shi. A\*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.
- [23] Josh Petty. What is 3d modeling & what's it used for. *Concept Art Empire*, 2018.

- [24] Dejan V Vranic and Dietmar Saupe. *3D model retrieval*. PhD thesis, University of Leipzig PhD thesis, 2004.
- [25] Wikipedia. 3D modeling — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=3D%20modeling&oldid=1154073519>, 2023. [Online; accessed 11-May-2023].
- [26] Sammy Ekaran. When did 3d modeling start? a brief history, Oct 2022.
- [27] Wikipedia. List of 3D modeling software — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=List%20of%203D%20modeling%20software&oldid=1148166501>, 2023. [Online; accessed 11-May-2023].
- [28] Roger C. Schank. What is ai, anyway? *AI Magazine*, 8(4):59, Dec. 1987.
- [29] Pei Wang. What do you mean by.^il In *AGI*, volume 171, pages 362–373, 2008.
- [30] Dexter C Kozen and Dexter C Kozen. Depth-first and breadth-first search. *The design and analysis of algorithms*, pages 19–24, 1992.
- [31] Adeel Javaid. Understanding dijkstra's algorithm. Available at SSRN 2340905, 2013.
- [32] Kurt Mehlhorn, Peter Sanders, and Peter Sanders. *Algorithms and data structures: The basic toolbox*, volume 55. Springer, 2008.
- [33] Ian Millington and John Funge. *Artificial intelligence for games*. CRC Press, 2009.
- [34] Marc HJ Romanycia and Francis Jeffry Pelletier. What is a heuristic? *Computational intelligence*, 1(1):47–58, 1985.
- [35] Richard E Korf. Linear-space best-first search. *Artificial intelligence*, 62(1):41–78, 1993.
- [36] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Computing Surveys (CSUR)*, 28(1):33–37, 1996.
- [37] Peter JM Van Laarhoven, Emile HL Aarts, Peter JM van Laarhoven, and Emile HL Aarts. *Simulated annealing*. Springer, 1987.
- [38] Hao Wu and Guohua Fan. An overview of tailoring strain delocalization for strength-ductility synergy. *Progress in Materials Science*, 113:100675, 2020.

- [39] Emile Aarts, Jan Korst, and Wil Michiels. Simulated annealing. *Search methodologies: introductory tutorials in optimization and decision support techniques*, pages 187–210, 2005.
- [40] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.
- [41] Melanie Mitchell, John Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing. *Advances in neural information processing systems*, 6, 1993.
- [42] Takumi Ohashi, Zaher Aghbari, and Akifumi Makinouchi. Hill-climbing algorithm for efficient color-based image segmentation. In *IASTED International Conference on Signal Processing, Pattern Recognition, and Applications*, pages 17–22, 2003.
- [43] Steven Hampson and Dennis Kibler. Plateaus and plateau search in boolean satisfiability problems: When to give up searching and start again. In *Workshop Notes: 2nd DIMACS Challenge*. Citeseer, 1993.
- [44] Matthias Biehl. *API Architecture*, volume 2. API-University Press, 2015.
- [45] Aram Cookson, Ryan DowlingSoka, and Clinton Crumpler. *Unreal Engine 4 Game Development in 24 Hours, Sams Teach Yourself*. Sams Publishing, 2016.
- [46] Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado. *UML 2 toolkit*. John Wiley & Sons, 2003.
- [47] Laurent Debrauwer and Fien Van der Heyde. *UML 2.5: iniciación, ejemplos y ejercicios corregidos*. Ediciones ENI, 2016.
- [48] Balázs Gregorics, Tibor Gregorics, Gábor Ferenc Kovács, András Dobreff, and Gergely Dévai. Textual diagram layout language and visualization algorithm. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 196–205. IEEE, 2015.
- [49] Siméon-Denis Poisson. *Researches into the Probabilities of Judgements in Criminal and Civil Cases*. NG-Verlag (Viatcheslav Demidov Inhaber), 2013.
- [50] James Andrew Storer. *An introduction to data structures and algorithms*. Springer Science & Business Media, 2001.

- [51] Chris Conlan. *The blender python API: Precision 3D modeling and add-on development*. Apress, 2017.
- [52] Romain Caudron and Pierre-Armand Nicq. *Blender 3D By Example*. Packt Publishing Ltd, 2015.
- [53] Victoria Horner and Andrew Whiten. Causal knowledge and imitation/emulation switching in chimpanzees (*pan troglodytes*) and children (*homo sapiens*). *Animal cognition*, 8:164–181, 2005.
- [54] Lisa Graham. Gestalt theory in interactive media design. *Journal of Humanities & Social Sciences*, 2(1), 2008.

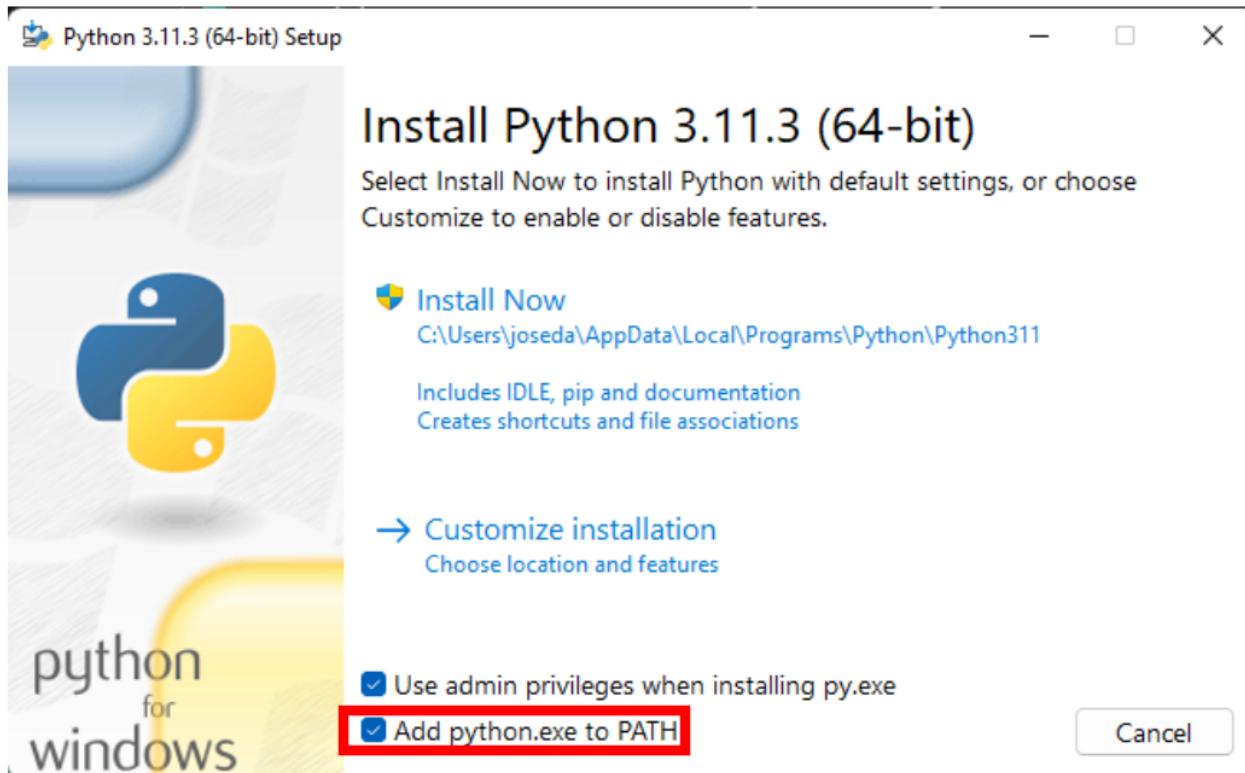
## Apéndice A

### Notas de instalación del repositorio

# TFG - Distribution of Custom Props in Blender | SurfaceSpray

## ▼ 1. Instalación Pseudo-automática

1. Es necesario tener instalado Python (para asegurar, instalarlo de todas formas, pinchando [aquí](#)) y añadido al PATH (opción en el instalador de Python) ↓↓↓.



# NOTA: En caso de instalarlo, esperar a que se instale completamente antes de proseguir.

El add-on se encuentra en el apartado [Releases](#) a la derecha de Github.

2. Descargar de la **ÚLTIMA Release** el auxiliar **PreparerSurfaceSpray.zip**, y abrir Blender en modo Administrador (click derecho sobre el programa y Ejecutar como administrador ).

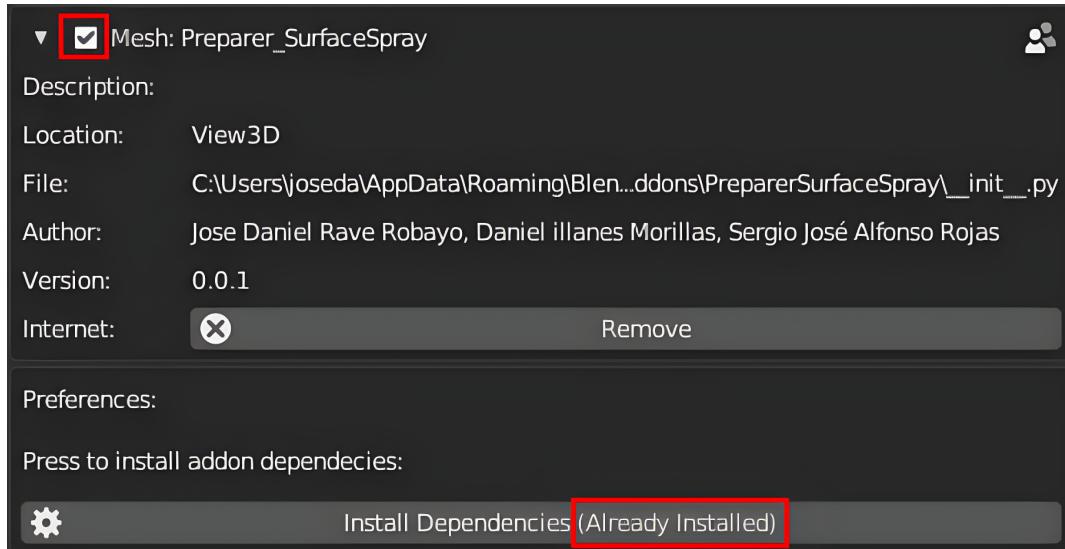
**IMPORTANTE:** En caso de tener Blender desde Steam, ir a `C:\Program Files (x86)\Steam\steamapps\common\Blender`, y abrir el ejecutable .exe

Para instalar el add-on ir a *Edit -> Preferencias ->* pestaña *Add-ons*.

3. Presionar el botón superior derecho **Install** y buscar el **.zip** del add-on y clickar **Install Add-on** .

Se deben esperar unos segundos a que se registre y aparezca en el buscador. En caso de que no aparezca, buscar "PreparerSurfaceSpray" y activarlo.

4. Posteriormente desplegar la pestaña del add-on y aparecerá un botón de **Install Dependencies** . Se debe presionar y esperar a que salga el mensaje (**Already Installed**). Es normal si el programa *No Responde* debido a la instalación.



**IMPORTANTE:** En caso que salte un error al presinar el botón, probar a volver a abrir Blender en **Modo Administrador**, o verificar que python está añadido al **PATH** (Environment Variable).

5. Una vez hecho esto, realizar la misma instalación con el add-on **SurfaceSpray.zip** de la **ÚLTIMA Release**, obviando la parte de instalación de dependencias. Una vez

activado, salir del **menú de add-ons**.

Presionar la tecla **N** sobre el 3D Viewport para mostrar el panel derecho de opciones. Se econtrará la pestaña **SurfaceSpray**.

**¡¡A disfrutar!!**

## ▼ 2. Instalación Manual

Es necesario instalar aima (y tener instalado Python). Para ello ejecutar el siguiente comando desde cualquier cmd .

```
pip install aima3
```

Se habrá añadido una carpeta llamada "*aima3*" al directorio *site-packages* (de la carpeta *python*) el cual se encuentra en la carpeta fuente de **Python**.

Ejecutando el siguiente código en una cmd, mostrará la localización del ejecutable python.

```
where python
```

Por ejemplo, debería estar encontrarse en una ruta similar a:

```
"C:\Users\user_name\AppData\Local\Programs\Python\Python310\Lib\site-packages"
```

A continuación, en dicha ruta copiar la carpeta *aima3* a la carpeta de python que utiliza Blender: debería ser la carpeta de Blender cuya ruta sería tal que:

```
Blender-X.X/python/lib/site-packages
```

Si sólo se tiene una versión de Blender instalada, y dicha carpeta coincide con la versión, ya no hay que hacer nada más.

En caso de no saber dónde se encuentra la ruta de python que usa Blender, o si se tiene más de una versión de Blender instalada, realizar los siguientes pasos:

- Abrir la versión de Blender deseada.
- Abrir la pestaña *Scripting* (barra de pestañas superior).
- Crear un nuevo fichero usando el botón + *New*.
- Copiar y pegar el siguiente código, y ejecutarlo dándole al botón RUN situado arriba a la derecha o usar el atajo de teclado *Alt + P*.

```
import site

usersitepackagespath = site.getsitepackages()

print("Path: ", usersitepackagespath)
```

Abrir *Toggle System Console* desde arriba a la izquierda *Window -> Toggle System Console*, para así poder ver el texto impreso por el código anterior. El texto debería mostrar la carpeta de python que ésta versión de Blender usa.

## Apéndice B

### Enlaces relacionados con el proyecto

- **Repositorio.**

<https://github.com/SergioJAlfonso/TFG-Distribution-System-of-Custom-Props-in-Blender>

- **Cuestionario de las pruebas.**

Primer Experimento:

<https://forms.gle/qtfZGgn16kMj5Bo3A>

Segundo Experimento:

<https://forms.gle/QsXqCKkvok8WBYS7A>

- **Videos de las pruebas.**

[https://drive.google.com/drive/folders/1G\\_Z5JzIuHIHB4euuLeSebnfZhzvJVy6T?usp=share\\_link](https://drive.google.com/drive/folders/1G_Z5JzIuHIHB4euuLeSebnfZhzvJVy6T?usp=share_link)

## Apéndice C

### Documentos del proceso de evaluación

#### C.1. Cuestionario inicial del primer experimento

# Surface Spray | Pruebas con Usuarios

Este cuestionario es meramente para contrastar los resultados de las pruebas. No nos hace falta tu e-mail.

\* Indica que la pregunta es obligatoria

---

1. Edad \*

---

2. Género \*

Marca solo un óvalo.

Hombre

Mujer

Prefiero no decirlo

Otro:

---

3. Ocupación \*

Marca solo un óvalo.

Opción 1

4. ¿Cuánta experiencia tienes con programas de modelado 3D? \*

Marca solo un óvalo.

Poca

Media

Mucha

Avanzado

5. ¿Conoces **Blender**? \*

Marca solo un óvalo.

 Sí NoManejo de Blender6. ¿Qué tal te manejas con **Blender**? \*

Marca solo un óvalo.

Principiante

1

2

3

4

5

Experto

---

Este contenido no ha sido creado ni aprobado por Google.

Google Formularios

## C.2. Cuestionario inicial del segundo experimento

# Surface Spray | Pruebas con Usuarios

Este cuestionario es meramente para contrastar los resultados de las pruebas. No nos hace falta tu e-mail.

\* Indica que la pregunta es obligatoria

---

1. Edad \*

---

2. Género \*

Marca solo un óvalo.

Hombre

Mujer

Prefiero no responder

Otro:

---

3. Ocupación \*

---

4. ¿Cuánta experiencia tienes con programas de modelado 3D? \*

Marca solo un óvalo.

Ninguna

Poca

Media

Mucha

Avanzada

Manejo de Blender

5. ¿Conoces **Blender**? \*

Marca solo un óvalo.

- Sí  
 No

6. ¿Cuánto tiempo de experiencia tienes con Blender? \*

Marca solo un óvalo.

- Nada  
 Menos de 1 año  
 1 año  
 2 años  
 3 años  
 4 años  
 Más de 4 años

7. Comentarios adicionales

---

---

---

---

---

---

Este contenido no ha sido creado ni aprobado por Google.

Google Formularios

1	Marca temporal	A	Edad	B	Género	C	Ocupación	D	¿Cuánta experiencia tienes con programas de modelado 3D?	E	¿Conoces Blender?	F	¿Cuánta experiencia tienes con Blender?	G	¿Conoces Blender?	H	Comentarios adicionales
2	2023/04/26 13:39:02 p. m.	EET	21	Mujer			Estudiante			Media	Si			Baja			Llevo usando Blender varios años, pero hace más de medio año que no lo uso.
3	2023/04/26 13:39:41 p. m.	EET	21	Mujer			Estudiante			Media	Si			Media			
4	2023/04/26 11:12:23 p. m.	EET	21	Hombre			Estudiante			Media	Si			Media			
5	2023/04/26 14:40 p. m.	EET	21	Hombre			Estudiante			Media	Si			Media			
6	2023/04/27 20:09:56 p. m.	EET	21	Mujer			Estudiante			Media	Si			Baja			
7	2023/04/27 20:09:56 p. m.	EET	21	Mujer													
8	2023/05/22 00:02:07 p. m.	EEST	21	Hombre			Estudiante			Media	Si			2 años			
9	2023/05/22 00:12:18 p. m.	EEST	22	Hombre			Estudiante			Avanzada	Si			Más de 4 años			
10	2023/05/22 03:27 p. m.	EEST	22	Hombre			Estudiante			Media	Si			Menos de 1 año			
11	2023/05/22 15:25 p. m.	EEST	24	Hombre			Desarrollador			Media	Si			Menos de 1 año			
12	2023/05/22 06:11 p. m.	EEST	22	No-bierno			Estudiante			Poca	Si			1 año			
13	2023/05/22 05:33 p. m.	EEST	25	Maier			Desarrolladora			Media	Si			Nada			
14	2023/05/23 00:07:03 p. m.	EEST	21	Hombre			Estudiante			Poca	Si			1 año			
15	2023/05/23 00:07:47 p. m.	EEST	22	Hombre			Estudiante			Media	Si			Menos de 1 año			
16	2023/05/23 00:07:57 p. m.	EEST	21	No-bierno			Estudiante			Mucha	Si			2 años			
17	2023/05/23 10:03:35 p. m.	EEST	22	Hombre			Estudiante y novelista			Poca	Si			1 año			

**Figura C.1:** Respuestas a las encuestas de ambos experimentos de la fase de evaluación.