



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Algoritmos y mazmorras
Documentación Técnica**



Presentado por Elsa Tolín Carrasco
en Universidad de Burgos — 7 de julio de 2024
Tutor: Jesús Alonso Abad y José Manuel
Galán Ordax

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	iv
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	5
Apéndice B Especificación de Requisitos	13
B.1. Introducción	13
B.2. Objetivos generales	13
B.3. Catalogo de requisitos	14
B.4. Especificación de requisitos	15
Apéndice C Especificación de diseño	21
C.1. Introducción	21
C.2. Diseño de datos	21
C.3. Diseño procedimental	23
C.4. Diseño arquitectónico	25
Apéndice D Documentación técnica de programación	31
D.1. Introducción	31
D.2. Manual del programador	31
D.3. Compilación, instalación y ejecución del proyecto	33

D.4. Pruebas del sistema	38
Apéndice E Documentación de usuario	43
E.1. Introducción	43
E.2. Requisitos de usuarios	43
E.3. Instalación	44
E.4. Manual del usuario	44
Apéndice F Sostenibilización curricular	49
F.1. Introducción	49
F.2. Integración Transversal de la Sostenibilidad	49
F.3. Desarrollo de Competencias para la Sostenibilidad	49
F.4. Fomento de la Investigación y Docencia Sostenible	50
F.5. Sensibilización y Participación de la Comunidad Universitaria	50
F.6. Establecimiento de Mecanismos de Evaluación y Mejora Continua	50
F.7. Creación de Redes y Plataformas de Colaboración	51
Bibliografía	53

Índice de figuras

B.1. Caso de uso de la aplicación	15
C.1. Diagrama del nodo DungeonResponse	22
C.2. Diagrama de secuencias de un laberinto nuevo.	24
C.3. Diagrama de secuencias de un laberinto ya almacenado.	25
C.4. Arquitectura cliente-servidor.	26
C.5. Tecnologías empleadas en la arquitectura	27
C.6. Patrón Modelo-Vista-Controlador	28
D.1. Pestaña de instalación de Unity en UnityHub.	34
D.2. Pestaña de instalación de nueva versión en Unity.	35
D.3. Ventana de docker para habilitar WSL. Imagen extraída de https://learn.microsoft.com/es-es/windows/wsl/tutorials/ wsl-containers	36
D.4. Pestaña de archivos en Visual Studio Code	37
D.5. Pestaña extensiones en Visual Studio Code	39
D.6. Pestaña de la extensión Dev Containers en Visual Studio Code	40
D.7. Notificación de reapertura de proyecto como Dev Container	40
D.8. Botón para ejecutar la apertura de una ventana remota	40
D.9. Menú de ejecución de ventana remota	41
D.10. Barra de opciones de Visual Studio Code	41
E.1. Carpeta del ejecutable de Unity.	44
E.2. Captura del menú principal del juego.	45
E.3. Captura del menú de selección de algoritmo.	46
E.4. Captura de la interfaz para introducir los parámetros del laberinto.	46
E.5. Captura del laberinto generado.	47

Índice de tablas

A.1. Coste anual por recursos	6
A.2. Coste total del proyecto	8
A.3. Dependencias de Python y sus licencias	10
B.1. CU-1 Inicio de aplicación	16
B.2. CU-2 Selección del algoritmo	16
B.3. CU-3 Parametrizar el laberinto	17
B.4. CU-4 Navegación del laberinto	18
B.5. CU-5 Abandonar la aplicación	19

Apéndice A

Plan de Proyecto Software

A.1. Introducción

El desarrollo del presente proyecto se ha extendido durante 3 años por motivos personales, pero el tiempo de desarrollo efectivo es menor. Es por este motivo que, para desarrollar la planificación se va a tener en cuenta el tiempo productivo.

Para el desarrollo de este proyecto se ha hecho una planificación utilizando la metodología Scrum. Se ha dividido el trabajo por sprints y para poder llevar un seguimiento de las tareas a realizar, se ha hecho uso del método Kanban. Para poder llevar a cabo esta metodología, se ha hecho uso de la aplicación web Jira. En el método Kanban se utiliza un tablón dividido por varias columnas, cada una especifica un estado de la tarea, y se va actualizando en este tablero según van progresando las tareas.

A.2. Planificación temporal

Por lo mencionado anteriormente, aunque no haya sido el tiempo real de desarrollo, se va a dividir la planificación de sprints efectivos de tres semanas.

Fases del desarrollo

1. Pruebas de concepto
2. Primera Fase

- a) Investigación del motor
- b) Investigación de algoritmos de generación procedural
- c) Implementación en el motor

3. Segunda Fase

- a) Preparación del servidor
- b) Preparación del front-end con Unity
- c) Preparación de ejecutable
- d) Tests
- e) Documentación

Sprint 1

Durante este sprint, se llevo a cabo una investigación de las posibles herramientas y lenguajes de programación que se podrían utilizar para desarrollar la idea del proyecto. Por ello se realizaron distintas pruebas de proyecto para comprobar qué problemas y qué ventajas se podrían encontrar durante el desarrollo.

- Java: Construcción de un prototipo desde cero en java con el algoritmo de Wilson
- C# y .NET: Construcción de un prototipo desde cero con C# y .NET.

Sprint 2

Durante este sprint, tras la investigación y las pruebas de concepto se comenzó a investigar la posibilidad de usar un motor gráfico como Unity.

- Aprendizaje sobre Unity: Se realizó un aprendizaje de este motor ya que es complejo y se carecía de experiencia previa con esta herramienta.
- Preparación del entorno: Se creó el proyecto en el motor y se preparó la primera escena

Sprint 3

En este sprint se trabajó en la estructura y en el algoritmo para poder generar el laberinto. Se necesitó de mucha investigación para aprender sobre cómo realizar estos pasos en Unity.

- Desarrollo del la escena para el menú y para el juego
- Búsqueda de un algoritmo sencillo para generar el laberinto
- Implementación del algoritmo

Sprint 4

Durante el desarrollo de este sprint se evaluaron los siguientes pasos y se comenzó a la «Segunda Fase» mencionada anteriormente. Se busca un nuevo diseño y se investiga como construir un servidor para conectarlo con el videojuego.

- Aprendizaje sobre como preparar el servidor
- Creación de la estructura base del servidor
- Implementación básica del servidor
- Implementación de la adaptación del algoritmo de Prim

Sprint 5

Tras la preparación del diseño del servidor y comenzar con su implementación básica, este siguiente sprint tiene como foco implementar en Unity el videojuego. En esta segunda fase el papel de Unity cambia a ser el encargado de dibujar los laberintos. En este sprint también se continua desarrollando algoritmos.

- Creación de las escenas necesarias para la interfaz de usuario
- Implementación del sistema para dibujar el laberinto
- Implementación de la adaptación del algoritmo de Kruskal
- Implementación de un DevContainer y se añade Docker a la arquitectura
- Cambio de SQLite a MongoDB

Sprint 6

En este sprint se continuó con el desarrollo de algoritmos de generación procedural y se realizó un notebook de python para comparar sus tiempos de ejecución.

- Implementación de la adaptación del algoritmo de DFS
- Implementación de la adaptación del algoritmo del Autómata celular
- Implementación de la adaptación del algoritmo del Eller
- Implementación de la adaptación del algoritmo del Teselación
- Implementación de la adaptación del algoritmo del Aldous Broder
- Implementación de la adaptación del algoritmo del Árbol binario
- Desarrollo en notebook para obtener gráficas para comparar los tiempos de ejecución

Sprint 7

Tras realizar la implementación de los algoritmos se preparó el ejecutable y se realizaron pruebas para buscar errores, tras esto se arreglaron los errores encontrados.

- Preparación de ejecutables
- Solución de errores

Sprint 8

Una vez terminado el proyecto se comenzó a desarrollar la documentación del proyecto. Durante este sprint se desarrolló la memoria.

Sprint 9

Tras desarrollar la memoria se realizó la documentación de los anexos del proyecto.

A.3. Estudio de viabilidad

Se va a proceder a realizar el estudio de viabilidad del proyecto. Este estudio no es representativo de un estudio de viabilidad de un videojuego, esto es porque este proyecto se ha centrado en el desarrollo visto desde un programador, no se tienen en cuenta el desarrollo artístico ya que se ha hecho uso de los recursos de la comunidad gratuitos.

Viabilidad económica

En esta sección se va realizar un análisis de los costes financieros que supone el desarrollo del presente proyecto. Para realizar este estudio, aunque el tiempo real de desarrollo han sido tres años, se tendrá en cuenta el tiempo de desarrollo efectivo. En otras palabras, se va a estudiar la viabilidad económica para un periodo efectivo. Los cálculos se realizarán como si se tratase de una nueva empresa con un único empleado, durante un periodo de 27 semanas.

También hay que tener en cuenta que en el caso de Unity, si el último ejercicio fiscal de los ingresos fueron superiores a 100 000 dolares estadounidenses, se tiene la obligación de comprar la edición profesional de Unity [14]. Como no se comercializa el juego, no se necesita tener en cuenta estos gastos. También se va a presuponer que el proyecto es el primero desarrollado con Unity por lo que no hay precedentes que obliguen a comprar la licencia profesional de Unity.

Para poder valorar la viabilidad económica se van a evaluar dos aspectos: los costes y los beneficios. En este caso, se asumirá que el proyecto lo lleva a cabo una empresa con liquidez suficiente como para que no sea necesaria una financiación.

Costes por recursos

En primer lugar, se van a analizar los costes derivados de los recursos del *software* y *hardware* que se han utilizado durante el desarrollo.

Las licencias utilizadas han sido gratuitas, y se listarán más adelante en consideración de un desarrollo futuro en el que se requiera obtener un plan de pago.

Como ya se ha mencionado al inicio de esta sección, aunque la duración del proyecto han sido tres años, vamos a tener en cuenta el tiempo efectivo, por lo que estos cálculos se van a efectuar para un proyecto cuyo periodo

de desarrollo ha durado 27 semanas, que equivalen a, aproximadamente, 7 meses.

- **Hardware:**

Durante el desarrollo de esta aplicación se ha hecho uso de 1 ordenador portátil. El modelo es MSI Raider GE66 12UH-005ES, este equipo tiene actualmente un precio de 3 098,86 € y se espera amortizar en 5 años [11]. Su coste anual de amortización será:

$$\frac{3\,098,86 \text{ €}}{5 \text{ años}} \approx 619,77 \text{ €/año}$$

- **Software:**

Se ha utilizado una licencia de sistema operativo Windows 11 Pro. Tiene un valor de 199 € mediante medios oficiales [9]. Dado que tiene una vida útil de 4 años, se espera amortizar en este tiempo.

$$\frac{199,00 \text{ €}}{4 \text{ años}} \approx 49,75 \text{ €/año}$$

De esta forma, el coste anual derivado de recursos queda definido en la Tabla A.1.

Tabla A.1: Coste anual por recursos.

Recurso	Coste anual
Windows 11 Pro	49,75 €
Portátil	619,77 €
Unity	0,00 €
Total:	669,52 €

Costes de personal

Para simplificar las fórmulas a continuación, utilizaremos las siguientes abreviaturas:

- **SBA:** Salario Bruto Anual (jornada parcial)
- **SBM:** Salario Bruto Mensual

- **CASS:** Contribución Anual a la Seguridad Social
- **CMSS:** Contribución Mensual a la Seguridad Social
- **CP:** Coste en Personal

Al realizar estas estimaciones, se va a considerar una empresa pequeña de un único empleado, como se ha mencionado anteriormente. Se va a asumir también, que la duración ha sido de 7 meses efectivos y ha sido desarrollado por un empleado de categoría «Junior». El sueldo anual bruto promedio se encuentra alrededor de unos 25 000 € con una jornada completa [7]. Para formalizar sus condiciones, se va a considerar que es un contrato a 30 horas a la semana con 12 pagas anuales.

$$\text{SBA} = 25\,000 \text{ €} \cdot \frac{30 \text{ horas}}{40 \text{ horas}} = 18\,750 \text{ €}$$

Con esto obtenemos el salario bruto mensual:

$$\text{SBM} = \frac{18\,750 \text{ €}}{12 \text{ pagas}} = 1\,562,5 \text{ €}$$

Para obtener el coste en personal hay que sumarle el salario bruto del empleado, calculamos la contribución anual a la seguridad social [13] por parte de la empresa. Esta se compone de 5 costes:

1. Contigencias comunes: 23,60 %
2. Desempleo: 5,5 %
3. Fogasa: 0,2 %
4. Formación profesional 0,6 %
5. Mecanismo de Equidad Intergeneracional 0,58 %

Haciendo que la contribución sea un 30,48 % del total.

$$\text{CASS} = 25\,000 \text{ €} \times 0,3048 = 7\,620 \text{ €}$$

$$\text{CMSS} = \frac{7\,620 \text{ €}}{12} \approx 635 \text{ €/mes}$$

El coste en personal total sería el siguiente:

$$CP = SBM + CMSS$$

$$CP = 1\,562,5 \text{ €} + 635 \text{ €} = 2\,197,5 \text{ €}$$

Coste total

Considerando todos los gastos anteriores, durante un periodo de 7 meses, se calcula el coste total del proyecto (v. Tabla A.2).

$$CP (7 \text{ meses}) = 2\,197,5 \text{ €} \cdot 7 \text{ meses} = 15\,322,5 \text{ €}$$

Tabla A.2: Coste total del proyecto

Concepto	Coste
Personal	15\,322,5 €
<i>Hardware</i>	361,53 €
<i>Software</i>	29,02 €
Otros	0,00 €
Coste Total del proyecto:	15\,713,05 €

Viabilidad legal

En esta sección se analizará la viabilidad legal del proyecto, revisando las licencias asociadas a cada una de las dependencias utilizadas. Es crucial garantizar que todas las licencias sean compatibles con el uso previsto del software, especialmente cuando se busca elegir la más restrictiva posible para asegurar el cumplimiento legal.

Licencias de Unity

Para el desarrollo de este proyecto se han utilizado las siguientes licencias relacionadas con Unity:

- **Unity:** Licencia gratuita. Para más información, se puede consultar el siguiente enlace: <https://support.unity.com/hc/en-us/categories/201268913-Licenses>.

- **Elementos de interfaz gráfica de usuario:** Licencia para uso personal y comercial disponible en: <https://mandinhart.itch.io/garden-cozy-kit-uigui-buttons-and-icons>.

Dependencias de Python

Las dependencias de Python utilizadas en este proyecto y sus respectivas licencias¹ se detallan en la Tabla A.3. Nos servirán para poder determinar qué licencia debemos emplear en el repositorio.

Licencia del proyecto

La elección de la licencia más restrictiva es esencial para asegurar que el uso del *software* cumple con todas las restricciones y obligaciones legales. En el análisis, las licencias BSD-3-Clause y Apache 2.0 son las más restrictivas entre las utilizadas. Ambas requieren que se incluya una copia de la licencia original, manteniendo así el reconocimiento de los derechos de los autores originales. La licencia Apache 2.0 añade, además, una cláusula de patentes, lo que puede incrementar las restricciones de uso en comparación con la BSD-3-Clause [1, 6].

Por lo tanto, para maximizar la compatibilidad y asegurar el cumplimiento legal, se recomienda adoptar las prácticas y obligaciones definidas por la licencia Apache 2.0, dado que incluye las restricciones adicionales sobre patentes y proporciona una mayor cobertura en términos de cumplimiento legal.

Implicaciones legales y cumplimiento

Es fundamental que los desarrolladores comprendan las implicaciones legales de cada licencia utilizada en el proyecto. A continuación, se presentan algunas directrices para cumplir con las obligaciones legales al redistribuir o modificar el *software*:

- **Incluir una copia de la licencia.** Al distribuir el *software*, es obligatorio incluir una copia de la licencia elegida.
- **Mantener los avisos de derechos de autor.** No se deben eliminar ni modificar los avisos de derechos de autor presentes en el código fuente.

¹Es importante destacar que toda la información referente a las licencias de las dependencias de Python se puede consultar fácilmente en <https://pypi.org/>, lo cual facilita la verificación y actualización constante de las licencias utilizadas en el proyecto.

- **Proporcionar modificaciones bajo la misma licencia.** Si se realizan modificaciones a una dependencia con una licencia *copyleft*, como la Apache 2.0, es necesario redistribuir dichas modificaciones bajo la misma licencia.
- **Informar sobre las licencias aplicables.** La documentación del proyecto debe incluir información clara sobre las licencias aplicables a cada parte del *software*.

Siguiendo estas directrices, se asegura que el proyecto cumple con todas las obligaciones legales y se minimiza el riesgo de infracciones de licencias.

Tabla A.3: Dependencias de Python y sus licencias

Dependencia	Versión	Licencia
annotated-types	0.6.0	MIT License ^a
anyio	4.3.0	MIT License ^a
debugpy	1.8.1	MIT License ^a
executing	2.0.1	MIT License ^a
fastapi	0.110.1	MIT License ^a
h11	0.14.0	MIT License ^a
jedi	0.19.1	MIT License ^a
parso	0.8.4	MIT License ^a
platformdirs	4.2.0	MIT License ^a
pure-eval	0.2.2	MIT License ^a
pydantic	2.7.0	MIT License ^a
pydantic_core	2.18.1	MIT License ^a
six	1.16.0	MIT License ^a
stack-data	0.6.3	MIT License ^a
toml	0.10.2	MIT License ^a
asttokens	2.4.1	Apache License 2.0 ^b
beanie	1.25.0	Apache License 2.0 ^b
lazy-model	0.2.0	Apache License 2.0 ^b
motor	3.4.0	Apache License 2.0 ^b
packaging	24.0	Apache License 2.0 ^b y BSD-2-Clause License ^c
pymongo	4.6.3	Apache License 2.0 ^b
python-dateutil	2.9.0.post0	Apache License 2.0 ^b
tornado	6.4	Apache License 2.0 ^b

Dependencias de Python y sus licencias (continuación)

Dependencia	Versión	Licencia
click	8.1.7	BSD-3-Clause License ^d
comm	0.2.2	BSD-3-Clause License ^d
idna	3.7	BSD-3-Clause License ^d
ipykernel	6.29.4	BSD-3-Clause License ^d
ipython	8.23.0	BSD-3-Clause License ^d
jupyter_client	8.6.1	BSD-3-Clause License ^d
jupyter_core	5.7.2	BSD-3-Clause License ^d
matplotlib-inline	0.1.6	BSD-3-Clause License ^d
prompt-toolkit	3.0.43	BSD-3-Clause License ^d
psutil	5.9.8	BSD-3-Clause License ^d
pymq	25.1.2	BSD-3-Clause License ^d
starlette	0.37.2	BSD-3-Clause License ^d
traitlets	5.14.2	BSD-3-Clause License ^d
uvicorn	0.29.0	BSD-3-Clause License ^d
decorator	5.1.1	BSD-2-Clause License ^c
nest-asyncio	1.6.0	BSD-2-Clause License ^c
Pygments	2.17.2	BSD-2-Clause License ^c
dnspython	2.6.1	ISC License ^e
pexpect	4.9.0	ISC License ^e
ptyprocess	0.7.0	ISC License ^e
typing_extensions	4.11.0	Python Software Foundation License ^f
sniffio	1.3.1	Apache License 2.0 ^b y MIT License ^a

^a <https://mit-license.org/>^b <http://www.apache.org/licenses/>^c <https://opensource.org/license/bsd-2-clause>^d <https://opensource.org/license/BSD-3-clause>^e <https://www.isc.org/licenses/>^f <https://docs.python.org/3/license.html>

Apéndice B

Especificación de Requisitos

B.1. Introducción

Para poder llevar a cabo un proyecto de *software*, se necesitan tener requisitos especificados, con esto se podrá realizar el desarrollo, usando estos requisitos como guía.

En este apartado se van a abarcar las especificaciones de requisitos y los casos de uso que reflejarán el funcionamiento del producto de *software* desarrollado.

B.2. Objetivos generales

Este proyecto tiene como objetivo principal la generación de laberintos que puedan ser navegables por el usuario. Los objetivos para llevar a cabo el proyecto son los siguientes:

1. Obtener laberintos que sean navegables.
2. Utilizar varios algoritmos de generación procedural para generar los laberintos.
3. Usar la arquitectura cliente-servidor para poder almacenar datos del videojuego.
4. Utilizar el protocolo HTTP para que el servidor se comunique con Unity.
5. Calcular los tiempos de ejecución de estos algoritmos y compararlos.

6. Generar una imagen del contenido con Docker para que se pueda alojar fácilmente en un servidor externo.

B.3. Catalogo de requisitos

Requisitos funcionales

Los requisitos funcionales definen el comportamiento que se espera por la aplicación. Para este proyecto se han redactado los siguientes:

■ RF1 - Laberintos navegables

- **RF1.1:** El usuario puede navegar por el laberinto con el objeto «Player» utilizando las teclas W, A, S, D.
- **RF1.2:** El usuario colisiona con los muros, es decir, no puede atravesar paredes.
- **RF1.3:** El usuario puede tener acceso a la pantalla anterior con un botón de retorno.
- RF1.4:

■ RF2 - Gestión del menú principal

- **RF2.1:** El usuario puede navegar por el menú principal
- **RF2.2:** El usuario puede abandonar el juego.
- **RF2.3:** El usuario puede moverse a la siguiente pantalla a través del menú.

■ RF3 - Gestión del menú secundario

- **RF3.1:** El usuario puede navegar el menú secundario.
- **RF3.2:** El usuario puede seleccionar el algoritmo que desea usar para la generación del laberinto.
- **RF3.3:** El usuario debe poder acceso a la pantalla anterior con un botón de retorno.

■ RF4 - Gestión de los laberintos generados

- **RF4.1:** La aplicación puede almacenar los laberintos con sus dimensiones y semilla generados.

■ RF5 - Generación de laberintos parametrizables

- **RF5.1:** El usuario puede introducir el ancho del laberinto a generar si el algoritmo lo permite.
- **RF5.2:** El usuario puede introducir el largo del laberinto a generar si el algoritmo lo permite.
- **RF5.3:** El usuario puede introducir el número de iteraciones si el algoritmo lo permite.
- **RF5.4:** El usuario puede introducir la semilla de la generación procedural.

Requisitos no funcionales

Los requisitos no funcionales engloban aquellos que no especifican el funcionamiento de la aplicación, van a definir el comportamiento del producto.

- **RNF1 - Velocidad de respuesta**
 - **RNF1.1:** La carga del laberinto debe ser lo más rápida posible.
- **RNF2 - Transparencia**
 - **RNF2.1:** Los cambios futuros deberán ser transparentes al usuario, ya que no hay necesidad de que conozca cómo funciona internamente el programa
- **RNF3 - Usabilidad**
 - **RNF3.1:** La navegación por la aplicación debe de ser cómoda.
 - **RNF3.2:** La navegación por los laberintos debe de ser intuitiva.

B.4. Especificación de requisitos

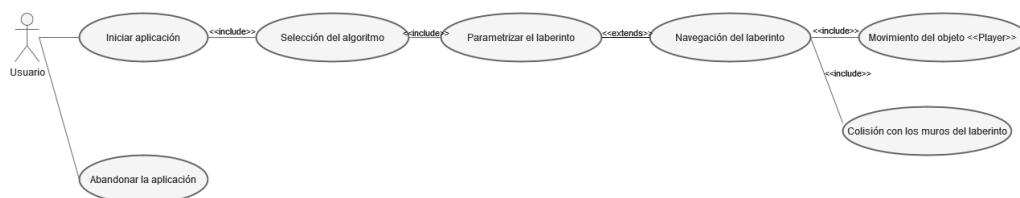


Figura B.1: Caso de uso de la aplicación.

CU-1	Iniciar aplicación
Versión	1.0
Autor	Elsa Tolín
Requisitos asociados	RF-2.1, RF-2.3
Descripción	Este caso de uso permite visualizar el menú principal y seleccionar el botón de arranque
Precondición	Arranque de la aplicación
Acciones	<ol style="list-style-type: none"> 1. Acceder a la página del menú principal 2. Pulsar botón
Postcondición	Redirección al menú secundario
Excepciones	
Importancia	Alta

Tabla B.1: CU-1 Inicio de aplicación.

CU-2	Selección del algoritmo
Versión	1.0
Autor	Elsa Tolín
Requisitos asociados	RF-3.1, RF-3.2
Descripción	El usuario puede seleccionar el algoritmo que desea probar en la aplicación
Precondición	Seleccionar el botón «comenzar» en la pantalla del menú principal
Acciones	<ol style="list-style-type: none"> 1. Acceder al menú secundario 2. Pulsar botón del algoritmo deseado
Postcondición	Redirección a la pantalla de generación de laberintos.
Excepciones	
Importancia	Alta
Nota	Este caso de uso se aplica para todos los algoritmos que puede seleccionar el usuario

Tabla B.2: CU-2 Selección del algoritmo.

CU-3	Parametrizar el laberinto
Versión	1.0
Autor	Elsa Tolín
Requisitos asociados	RF-5, RF-5.1, RF-5.2, RF-5.3, RF-5.4
Descripción	Se puede parametrizar el tamaño del laberinto, ya sea por ancho y largo o por número de iteraciones
Precondición	Seleccionar el botón del algoritmo que se va a probar
Acciones	<ol style="list-style-type: none"> 1. Acceder al menú secundario 2. Selección del botón según el algoritmo que se vaya a probar 3. Introducir ancho y largo en las cajas de texto 4. Introducir semilla (opcional) 5. Pulsar el botón de «Generar mazmorra»
Postcondición	Se genera el laberinto
Excepciones	En caso de que sea el algoritmo de teselación o de árbol binario se pedirán al usuario las iteraciones. Si los datos introducidos no son correctos pedirá al usuario volverlos a introducir. Si el campo de la semilla no se completa se utilizará una semilla generada por el programa.
Importancia	Alta

Tabla B.3: CU-3 Parametrizar el laberinto.

CU-4	Navegación del laberinto
Versión	1.0
Autor	Elsa Tolín
Requisitos asociados	RF-1, RF-1.1, RF-1.2
Descripción	El usuario se puede mover por el laberinto en tiempo real.
Precondición	Pulsar el botón de «Generar mazmorra»
Acciones	<ul style="list-style-type: none"> 1. El usuario pulsa una de las teclas (W A S D) se desplaza
Postcondición	<ul style="list-style-type: none"> 1. El usuario no ha atravesado ni se ha introducido en un muro 2. El usuario ha cambiado su posición dentro del laberinto
Excepciones	Colisiona con un muro
Importancia	Alta

Tabla B.4: CU-4 Navegación del laberinto

CU-5	Abandonar la aplicación
Versión	1.0
Autor	Elsa Tolín
Requisitos asociados	RF1.3, RF-2.2
Descripción	El usuario puede abandonar la aplicación y que la ventana se cierre automáticamente
Precondición	
Acciones	<ol style="list-style-type: none"> 1. El usuario se encuentra en la pantalla de exploración de laberinto 2. El usuario pulsa sobre el botón «volver» 3. El usuario se encuentra en la pantalla de selección de algoritmo 4. El usuario pulsa sobre el botón «volver» 5. El usuario se encuentra en la pantalla principal 6. El usuario pulsa sobre el botón «salir»
Postcondición	Cierre de la aplicación
Excepciones	
Importancia	Alta

Tabla B.5: CU-5 Abandonar la aplicación

Apéndice C

Especificación de diseño

C.1. Introducción

La especificación de diseño es un componente fundamental en la creación de sistemas robustos y escalables. En esta sección se detallará el diseño de los datos, secuencias y arquitectura del *software* desarrollado. Esto permite visualizar de forma sencilla la estructura y los datos que componen el proyecto.

C.2. Diseño de datos

En este proyecto se utiliza una base de datos NoSQL implementada mediante Beanie¹, un Object Document Mapper (ODM) para Python, que permite una interacción sencilla y eficiente con MongoDB. A continuación, se describen detalladamente las estructuras de datos empleadas y su implementación.

Modelo de datos

El modelo de datos se define mediante la clase `DungeonResponse`, la cual hereda de `Document` y `BaseModel`. Esta combinación permite utilizar las ventajas de un ODM y las validaciones de datos ofrecidas por Pydantic. Para utilizar este modelo de datos en nuestra aplicación, es necesario seguir una serie de pasos. Primero, se debe conectar a la base de datos MongoDB y

¹La documentación oficial se puede consultar en el siguiente enlace: <https://beanie-odm.dev/>

luego realizar las operaciones CRUD (*e.g.*, crear, leer, actualizar o eliminar) sobre los documentos `DungeonResponse`.

A continuación, se presenta la definición de la clase que modela el documento.

```

1 from beanie import Document
2 from typing import List, Dict, Any
3 from pydantic import BaseModel
4
5 class DungeonResponse(Document, BaseModel):
6     algorithm: str
7     seed: int
8     parameters: Dict[str, Any]
9     maze: List[List[int]]

```

Descripción de los campos

- **algorithm**: este campo de tipo `str` almacena el nombre del algoritmo utilizado para generar el laberinto. Es fundamental para identificar el método específico de generación empleado.
- **seed**: de tipo `int`, este campo guarda la semilla utilizada para la generación del laberinto. El uso de una semilla asegura la reproducibilidad de los laberintos generados.
- **parameters**: este es un diccionario (`Dict[str, Any]`) que contiene parámetros adicionales utilizados por el algoritmo de generación. La flexibilidad de este campo permite adaptarse a diversas configuraciones algorítmicas.
- **maze**: este campo es una lista de listas de enteros (`List[List[int]]`) que representa la estructura del laberinto generado, donde cada entero indica el tipo de celda correspondiente.

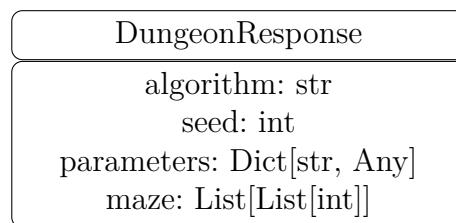


Figura C.1: Diagrama del nodo `DungeonResponse`

Ventajas del diseño NoSQL

El uso de una base de datos NoSQL, específicamente MongoDB, proporciona diversas ventajas en este contexto, a saber:

- **Flexibilidad en el esquema.** La estructura flexible de documentos en MongoDB permite modificar el modelo de datos sin necesidad de migraciones complejas.
- **Escalabilidad.** MongoDB está diseñado para manejar grandes volúmenes de datos y proporciona escalabilidad horizontal mediante la partición de datos (*sharding*).
- **Consultas rápidas.** Las consultas y agregaciones en MongoDB son rápidas y eficientes, lo cual es crucial para aplicaciones que requieren respuestas en tiempo real.

C.3. Diseño procedimental

En esta sección se describen los procedimientos utilizados en el proyecto para controlar el flujo de información que hay entre el usuario y el proyecto. Para poder ilustrarlos, se diseñan varios diagramas de secuencia siguiendo la normativa UML.

En primer lugar, se va a analizar el flujo de información que se da cuando se genera un laberinto nuevo; es decir, cuando el laberinto no está inicialmente almacenado en la base de datos del servidor. Corresponde con la figura C.2.

En segundo lugar, se va a analizar el caso en el que el laberinto sí se encuentra ya generado en la base de datos. Corresponde con la figura C.3.

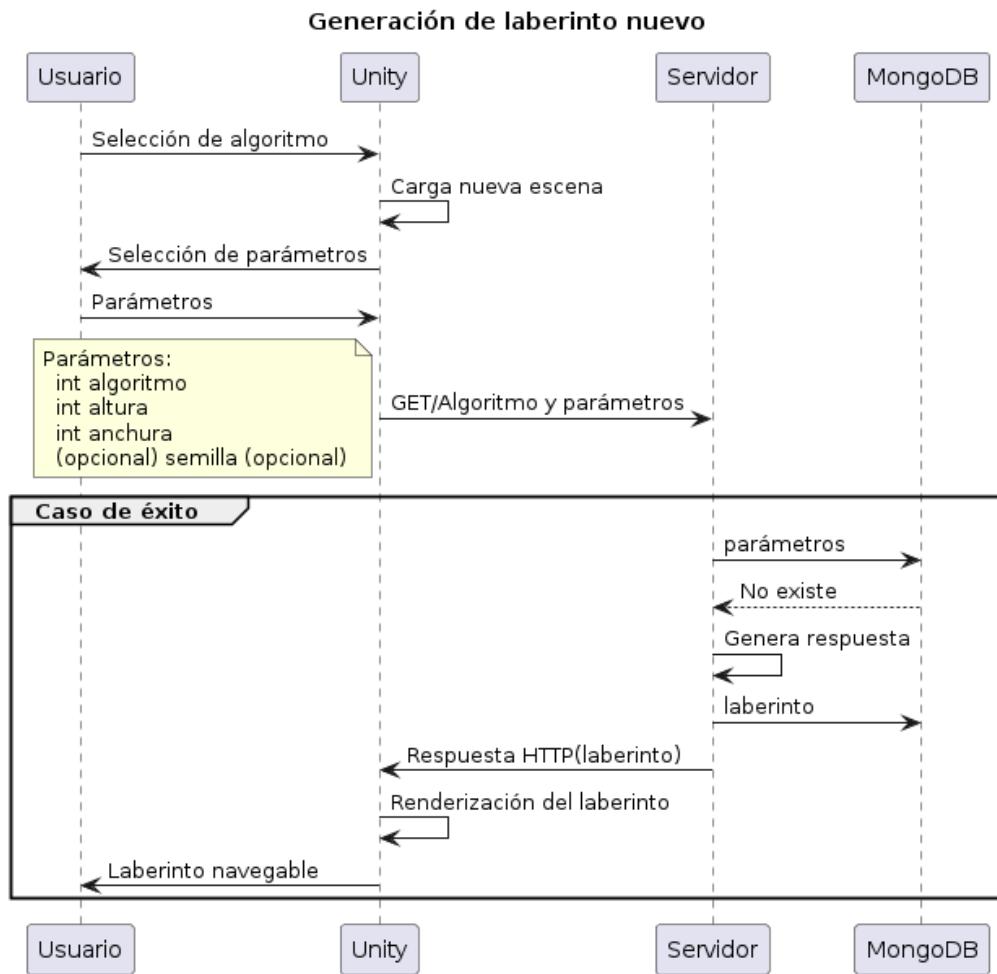


Figura C.2: Diagrama de secuencias de un laberinto nuevo.

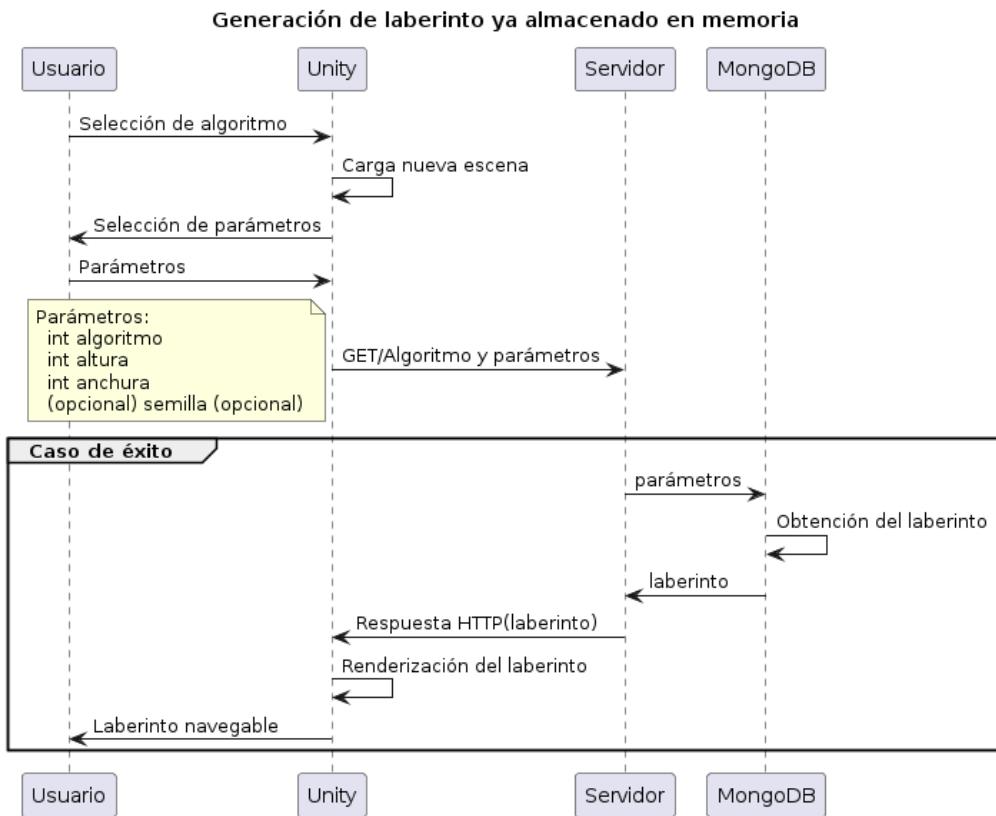


Figura C.3: Diagrama de secuencias de un laberinto ya almacenado.

C.4. Diseño arquitectónico

El diseño arquitectónico de la aplicación se basa en una arquitectura cliente-servidor, donde el cliente está desarrollado en Unity, el servidor utiliza FastAPI y la base de datos es MongoDB. Este enfoque permite una comunicación eficiente y estructurada entre los componentes, asegurando un flujo de datos coherente y una gestión centralizada de la lógica de negocio.

Arquitectura cliente-servidor

La arquitectura cliente-servidor empleada se divide en dos componentes principales:

- **Cliente.** La interfaz gráfica y la lógica de presentación de la aplicación se desarrollan en Unity. Este componente es responsable de interactuar con el usuario final y de enviar solicitudes al servidor.
- **Servidor.** Este componente maneja las solicitudes del cliente, procesa la lógica de negocio y realiza operaciones sobre la base de datos. FastAPI, un *framework* web moderno y rápido para Python, se utiliza para construir las API que facilitan la comunicación entre el cliente y el servidor. Por otro lado, en el lado del servidor localizamos también la base de datos MongoDB, que se utiliza para almacenar de manera eficiente los datos necesarios para la aplicación.

Esta arquitectura vista en conjunto no es monolítica, pero si se miran por separado cliente y servidor, cada uno de ellos son dos monolitos.

A continuación, en la Figura C.4, se presenta un diagrama que ilustra la arquitectura cliente-servidor.

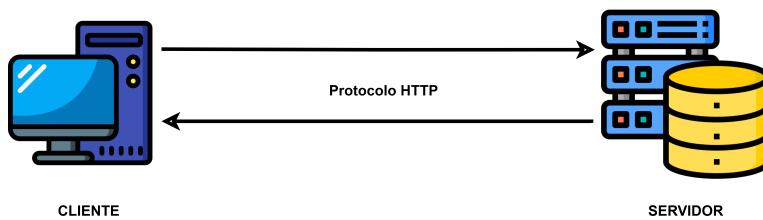


Figura C.4: Arquitectura cliente-servidor².

FastAPI y REST

FastAPI es un *framework* web moderno para Python que permite la creación rápida y eficiente de API RESTful. Este *framework* es conocido por su alto rendimiento y facilidad de uso, permitiendo a los desarrolladores definir y gestionar *endpoints* de manera declarativa.

REST

REST (Representational State Transfer) es un estilo arquitectónico para diseñar servicios web que utilizan HTTP como protocolo de comunicación. Los principios de REST incluyen:

²Diagrama creado empleando iconos vectoriales de libre acceso desde la página <https://www.flaticon.es/>

- **Statelessness.** Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud.
- **Resource-Based.** Los recursos (datos) son identificados por URLs y pueden ser manipulados usando los métodos HTTP estándar (GET, POST, PUT, DELETE).
- **Representation.** Los recursos son representados en formatos estándar como JSON o XML.

OpenAPI y Swagger

FastAPI soporta de manera nativa la generación de documentación OpenAPI y Swagger. OpenAPI es una especificación que define una forma estándar de describir y documentar las API RESTful. Swagger es un conjunto de herramientas para la creación de documentación interactiva de API basadas en la especificación OpenAPI.

FastAPI genera automáticamente la documentación de la API en formato OpenAPI, y proporciona una interfaz web interactiva mediante Swagger UI, donde los desarrolladores pueden explorar y probar los *endpoints* de la API.

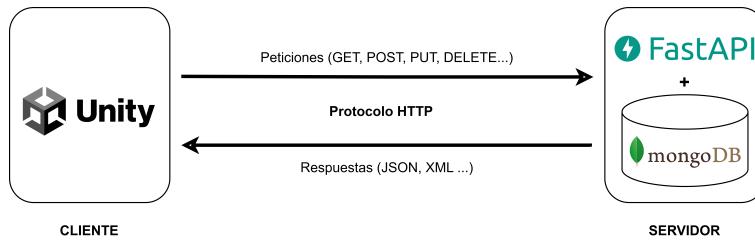


Figura C.5: Tecnologías empleadas en la arquitectura

Modelo-Vista-Controlador

El patrón de diseño Modelo-Vista-Controlador (MVC) se utiliza para estructurar la aplicación de manera que se separen claramente los intereses de datos, interfaz de usuario y control de flujo [3]. Este patrón se implementa de la siguiente manera:

- **Modelo:** representa los datos y la lógica de negocio de la aplicación. En este caso, incluye las clases y estructuras que interactúan con la base de datos MongoDB, como `DungeonResponse`.

- **Vista:** maneja la representación visual de los datos y la interacción con el usuario. Unity actúa como la vista en esta arquitectura, proporcionando una interfaz gráfica interactiva.
- **Controlador:** interpreta las entradas del usuario y las convierte en acciones sobre el modelo. FastAPI sirve como el controlador, gestionando las solicitudes del cliente, aplicando la lógica de negocio y respondiendo adecuadamente.

El siguiente diagrama muestra la implementación del patrón MVC en la arquitectura.

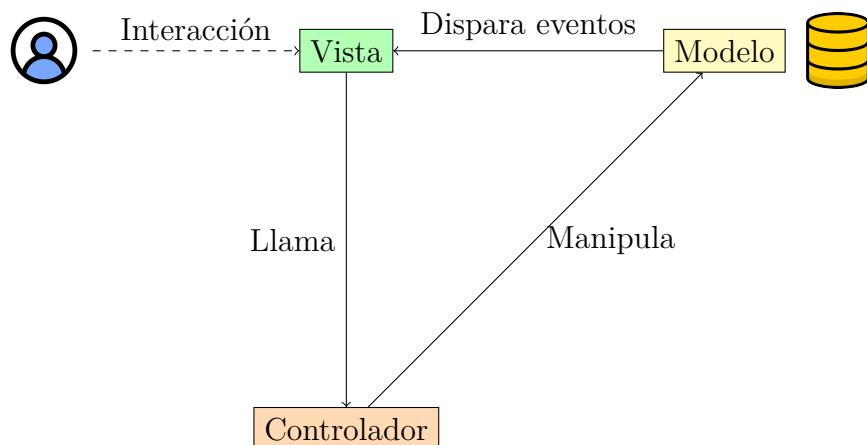


Figura C.6: Patrón Modelo-Vista-Controlador

Estrategia y Método Plantilla

En el diseño del servidor se utilizan los patrones de diseño Estrategia y Método Plantilla para mejorar tanto la modularidad como la reutilización del código. Estos patrones permiten definir algoritmos específicos de manera concreta, mientras que las partes comunes se establecen en una clase base, facilitando la extensión del diseño con nuevos algoritmos sin modificar el código existente.

El patrón de diseño Estrategia es un patrón de diseño de comportamiento que define una familia de algoritmos, hace que se coloque cada uno de estos algoritmos en una clase separada, es decir, se encapsulan, y se hacen intercambiables [4].

El patrón de diseño Método Plantilla es un patrón de diseño de comportamiento, este define el esqueleto de un algoritmo en la superclase pero también permite que las subclases sobrescriban pasos del algoritmo sin cambiar su estructura [5].

En el caso de este proyecto, cada algoritmo de generación procedural es una estrategia concreta, no se redefinen al completo si no que cada uno hace uso de las partes comunes que están definidas en la clase base y las partes específicas en las clases hijas. De esta forma se mejora la modularidad del código, se reutiliza y se reduce la duplicidad de código.

Al hacer uso de estos patrones se favorece el principio SOLID Open-Closed. Este propicia la extensión del diseño con nuevos algoritmos sin necesidad de modificar el código existente, haciendo un diseño abierto para extensión pero cerrado para la modificación [12]. También se ve favorecido el principio de sustitución de Liskov, este permite que cualquier algoritmo de una subclase pueda sustituir a otro de una superclase sin alterar el comportamiento del programa, así se garantiza que las estrategias sean consistentes [8].

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apartado se va a detallar la estructura del proyecto, las partes que lo componen y la función de cada una. El código final de la aplicación se encuentra dentro de `/SegundaFase`, el resto de directorios se mantienen como partes del repositorio(<https://github.com/etc99/algoritmosYmazmorras>) a modo de histórico de fases previas por las que ha pasado el proyecto antes de tomar la aproximación actual.

D.2. Manual del programador

El código del proyecto se divide entre dos partes principales; una para la aplicación de Unity, incluida en `/AlgoritmosYMazmorras`; y la parte del servidor de laberintos contenida en `/Mazmorras`.

`/AlgoritmosYMazmorras`

Este directorio contiene todo lo relativo a la aplicación de Unity. Sigue la estructura de un proyecto común de Unity.

`/Assets`

Contiene los recursos utilizados en el proyecto. En este directorio se encuentran los *scripts* que establecen los comportamiento de los distintos

componentes del juego. También contiene las escenas del juego, así como distintos recursos gráficos, de audio y de texturas utilizados en el juego. Además contiene los distintos Prefabs creados para el proyecto.

/Packages

Contiene los paquetes utilizados por Unity para el proyecto. Normalmente no hay necesidad de modificar nada de este directorio debido a que el propio Unity ya lo gestiona.

/Library

En este directorio se encuentran los archivos generados por Unity en la compilación del juego. Este directorio no es necesario que tenga seguimiento por Git.

/ProjectSettings

Este directorio contiene la configuración del proyecto, así como configuración específica del editor de Unity.

/Mazmorras

En este directorio está contenido el servidor de Python junto con los distintos generadores de laberintos, así como de su persistencia en la base de datos de MongoDB.

También contiene ficheros Dockerfile y docker-compose con las que levantar la aplicación en un entorno de contenedores de Docker.

/.devcontainer

Esta carpeta contiene los archivos de configuración para el uso de la tecnología de los devcontainers. Está compuesto por un fichero Dockerfile para configurar la imagen del servidor a utilizar en el contenedor de desarrollo, un fichero `docker-compose.yml` que contiene la configuración con la que lanzar los distintos contenedores de la aplicación para el desarrollo y un fichero `devcontainer.json` con configuración propia de la tecnología de los devcontainers.

/src

Dentro de este directorio está contenido el código que compone el servidor de laberintos.

El fichero `requirements.txt` contiene las dependencias de bibliotecas externas de nuestro proyecto de Python.

El servidor del backend junto a sus endpoints están declarados dentro del fichero `app.py`. Desde aquí será donde se manejarán las peticiones realizadas al servidor.

Para lanzar el servidor más cómodamente utilizando uvicorn se ha creado un *script* de nombre `main.py`.

/dungeon_api

Es la carpeta con los componentes principales del funcionamiento del servidor.

El directorio `/dungeon_controllers` contiene módulos que se encargan de la generación de los laberintos y su persistencia en la base de datos. Estos son utilizados en el manejo de peticiones del servidor.

El otro subdirectorio es `/dungeon_generators`, que contienen los generadores de laberintos de cada uno de los algoritmos implementados. Para implementarlos se ha recurrido a un enfoque de programación orientada a objetos para añadir funcionalidades que faciliten su serialización y configuración. Todos los generadores heredan de `DungeonBase`, que provee una interfaz común para los generadores. Está contenido en `dungeon_base.py`.

Para la gestión del almacenamiento en MongoDB de los laberintos se usan los modelos de `/models`. Los modelos definidos aquí dan la estructura que seguirán las entidades almacenadas en la base de datos.

D.3. Compilación, instalación y ejecución del proyecto

Para poder ejecutar este proyecto se tienen que preparar dos partes, el servidor y el cliente, en este caso Unity. En esta sección se va a explicar como preparar ambas partes para poder desarrollar el proyecto.

Instalación de Unity

Unity tiene soporte para ciertos sistemas operativos, entre ellos están

1. Windows 7,8,10,11
2. macOS X 10.13+
3. CentOS 7
4. Rocky
5. Ubuntu 18.04, 20.04, 22.04

Para poder realizar la instalación del motor gráfico, primero se necesita realizar la descarga de Unity Hub. En el caso de Windows y macOS se necesitaría proceder con los siguientes pasos:

1. Acceder a la web de Unity <https://unity.com/es/download>.
2. Desde esa URL se descarga Unity Hub.
3. Proceder con el proceso del instalador de la ventana de **setup**.

Al abrir Unity Hub por primera vez, pide el inicio de sesión con tu cuenta de Unity, si no se tiene una cuenta previamente se necesita crearla. Tras esto, en la ventana de unity hub, en el menú de la derecha aparecen distintos botones, entre ellos el botón de «Installs» donde aparecerá automáticamente el inicio de descarga de Unity como se muestra en la figura D.1.

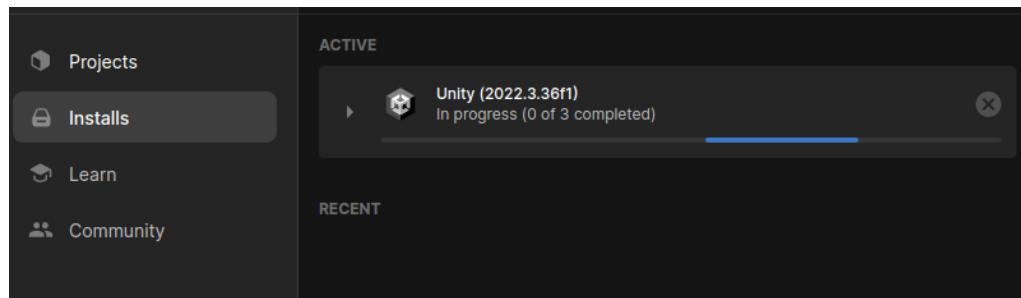


Figura D.1: Pestaña de instalación de Unity en UnityHub.

Es recomendable para poder visualizar y trabajar con los *scripts* de Unity instalar Visual Studio Community. Tras la instalación anterior se

puede realizar desde el Unity, Edit>Preferences y desde ahí se cambia la herramienta externa.

Tras esto solo quedaría exportar el proyecto a Unity. Unity va a pedir que se instale una versión previa y es porque este proyecto se hizo con la versión 2021.3.25f1.

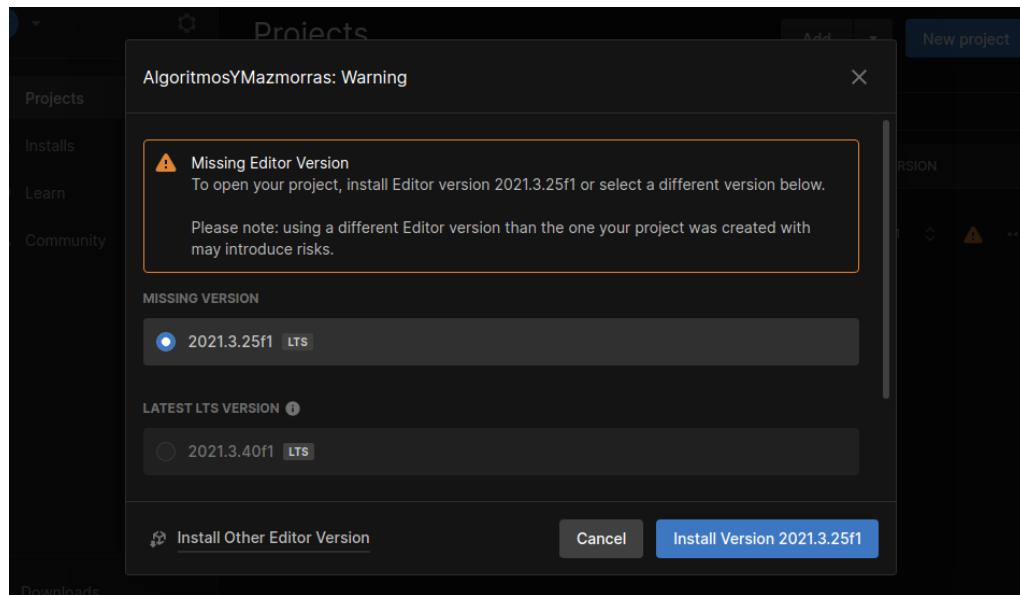


Figura D.2: Pestaña de instalación de nueva versión en Unity.

Instalación de Docker

Para instalar docker primero se necesita obtener de la página de descargas, se puede escoger la versión del siguiente enlace <https://docs.docker.com/desktop/release-notes/> y se obtendrá el instalador. Tras esto sólo quedaría seguir los pasos del instalador. Tras la instalación se necesita ejecutar con permisos administrador.

En máquinas con Windows, es necesario habilitar WSL [10], el subsistema Linux para Windows, para realizarlo se puede hacer desde Docker, como se muestra en la figura D.3.

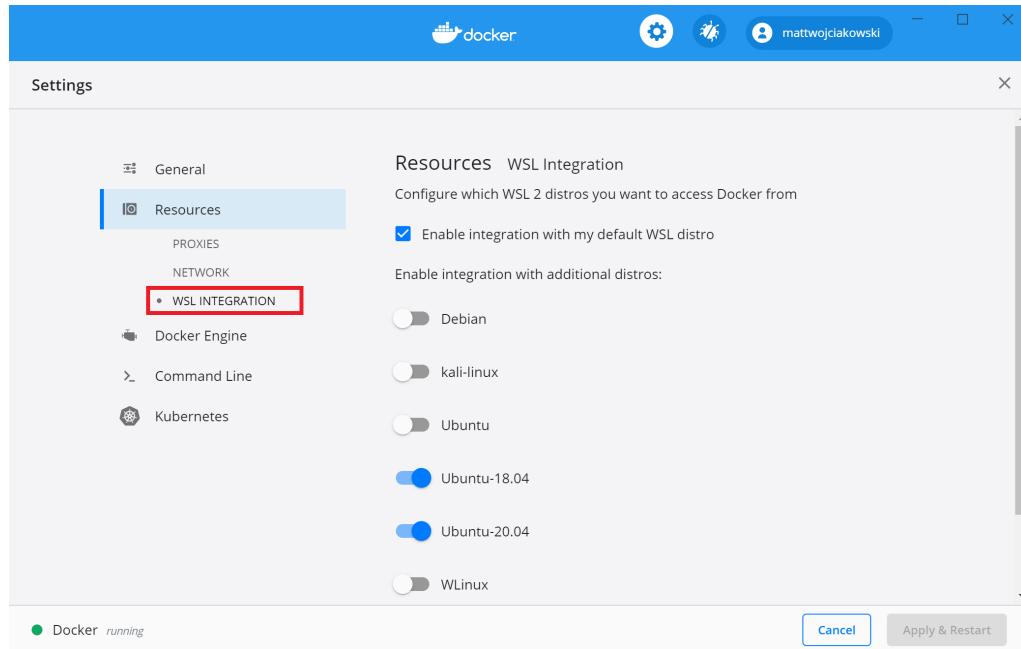


Figura D.3: Ventana de docker para habilitar WSL. Imagen extraída de <https://learn.microsoft.com/es-es/windows/wsl/tutorials/wsl-containers>

Preparación de DevContainer

Desde Visual Studio Code se puede abrir el contenido del proyecto correspondiente a la carpeta **/Mazmorras**. Para hacerlo en la esquina superior derecha como se muestra en la figura D.4.

Una vez abierto se necesita descargar la extensión **Dev Containers**. Para hacerlo en la barra de menú que se encuentra en el margen derecho se debe ir al botón que se muestra en la figura D.5. La extensión es la que se muestra en la figura D.6.

Al instalar la extensión, automáticamente aparece la notificación de la figura D.7, al pulsar en el botón de «Reopen in container» volverá a abrir el proyecto, esta vez desde el contenedor ya preparado.

Si no apareciese la notificación, se puede abrir con la extensión siguiendo los siguientes pasos. Primero se necesita pulsar el botón para abrir el proyecto desde una ventana remota, como se muestra en la figura D.8. Tras esto se abrirá el menú de selección como se muestra en la figura D.9. Se necesita

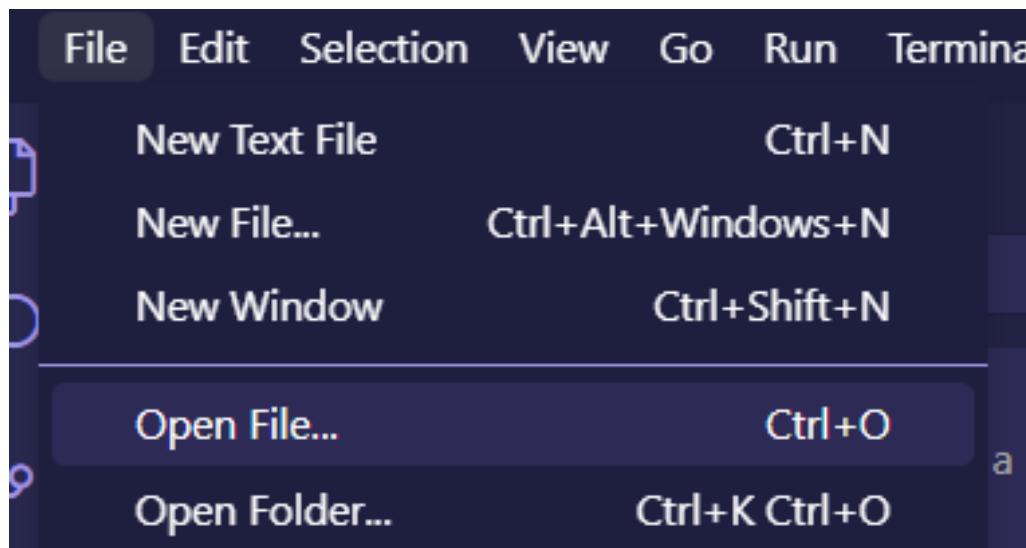


Figura D.4: Pestaña de archivos en Visual Studio Code

seleccionar la opción «Reopen in Container», al hacerlo se volverá a abrir el proyecto desde el contenedor.

Levantar el servidor

Para levantar el servidor se aconseja usar la terminal integrada de Visual Studio Code, esta se puede abrir desde la barra superior de la ventana como se muestra en la figura D.10.

El servidor de laberintos se lanza a través de un contenedor de Docker, siguiendo la configuración especificada en el Dockerfile. Para poder hacerlo tenemos que introducir en la terminal en la ruta del directorio del backend el siguiente comando:

```
1 docker compose up
```

Se lanzarán un contenedor con el servidor y otro de mongo, siguiendo la configuración del fichero Docker-compose.yml contenida en ese directorio.

Para poder parar el contenedor, desde el teclado se pulsarían los botones Ctrl+C y para borrar el contenedor se realiza con el siguiente comando:

```
1 docker compose down
```

D.4. Pruebas del sistema

Es necesario para asegurar una buena experiencia de usuario poner el sistema desarrollado a prueba. En el caso de este proyecto, está compuesto de dos partes que necesitan pruebas distintas.

Para realizar pruebas que comprueben el funcionamiento del servidor, se realizaron pruebas de peticiones predefinidas y respuestas esperadas, en este caso la herramienta elegida fue Postman¹.

Para comprobar el correcto funcionamiento del videojuego, se han preparado ejecutables que después se han testeado, a este proceso se le llama beta testeo. Se recomienda que este testeo lo realicen personas no familiarizadas con el proyecto, estas probarán el videojuego para poder encontrar problemas que el desarrollador no podría encontrar. El proceso de beta testeo fue esencial para obtener una experiencia de usuario adecuada.

¹Esta herramienta permite comprobar el correcto funcionamiento de una API a través de llamadas predefinidas, para más información <https://learning.postman.com/docs/introduction/overview/>

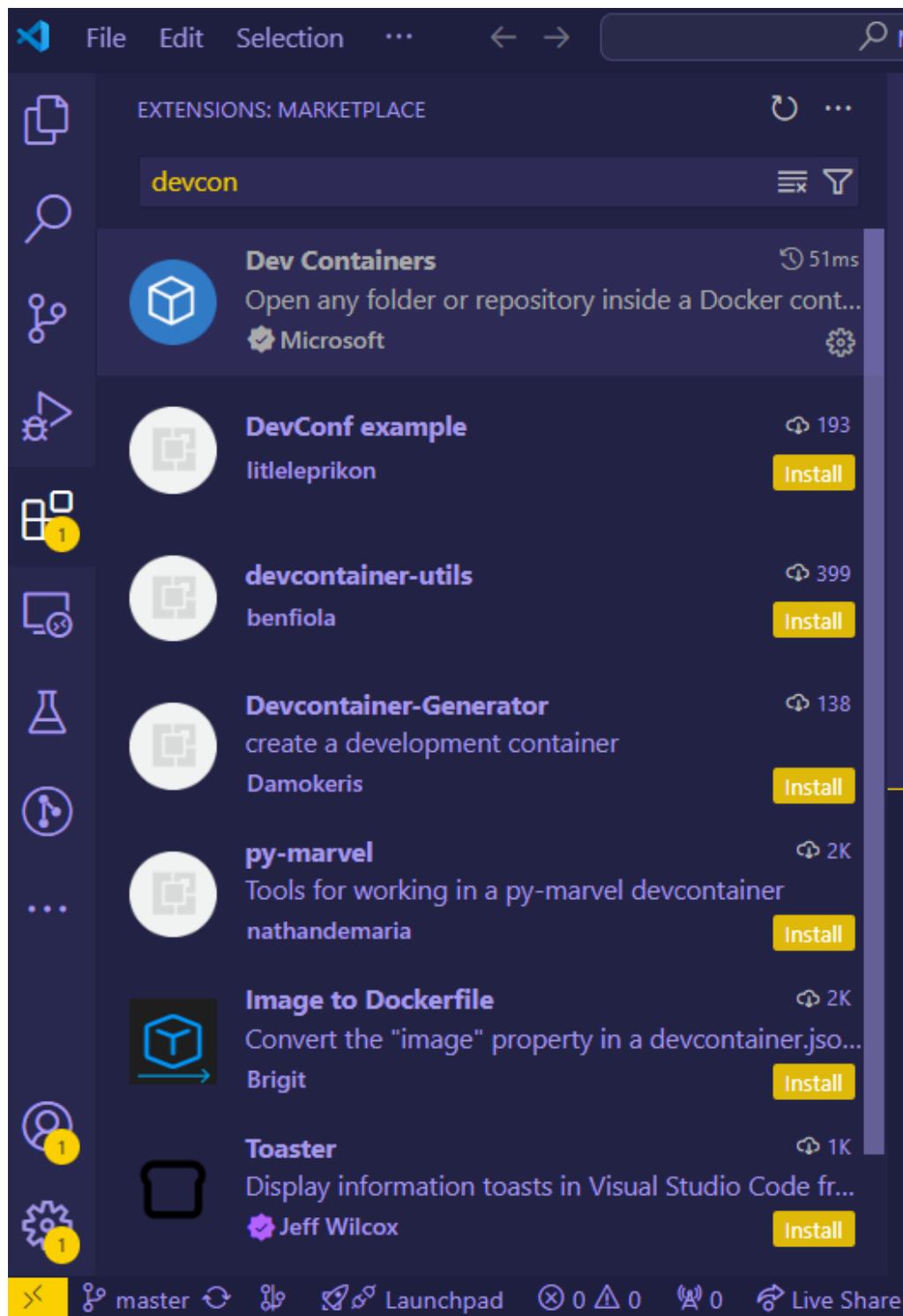


Figura D.5: Pestaña extensiones en Visual Studio Code

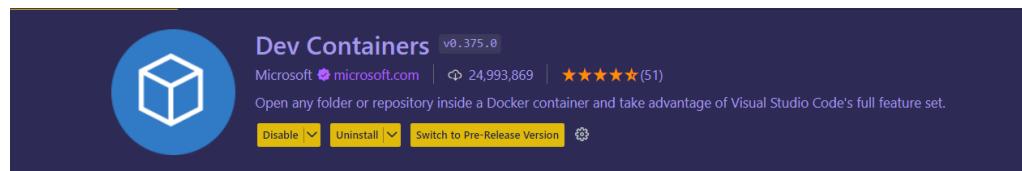


Figura D.6: Pestaña de la extensión Dev Containers en Visual Studio Code

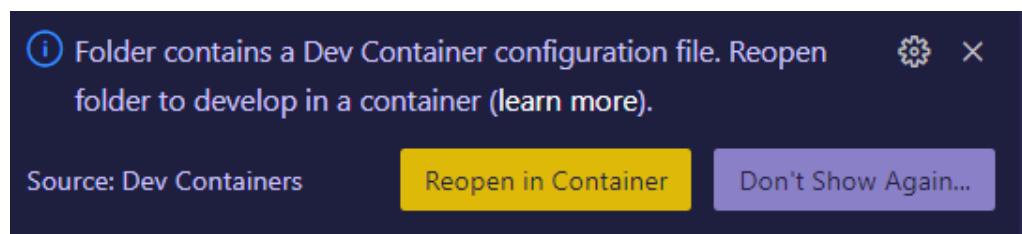


Figura D.7: Notificación de reapertura de proyecto como Dev Container

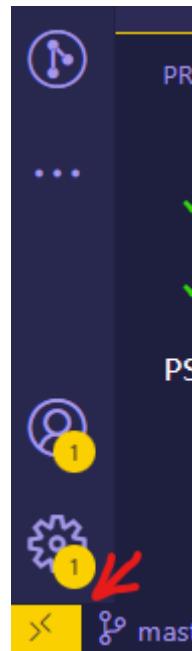


Figura D.8: Botón para ejecutar la apertura de una ventana remota

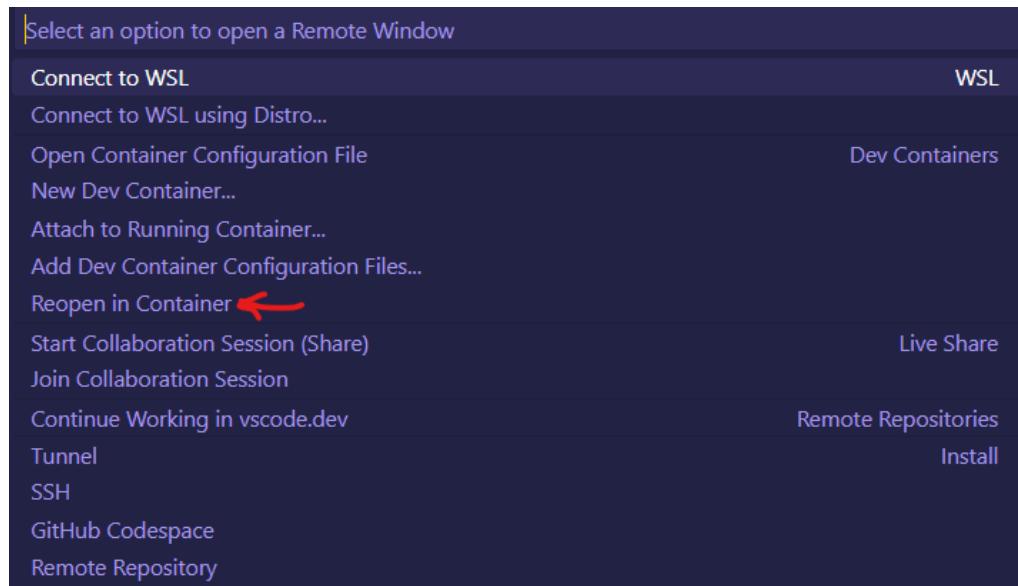


Figura D.9: Menú de ejecución de ventana remota

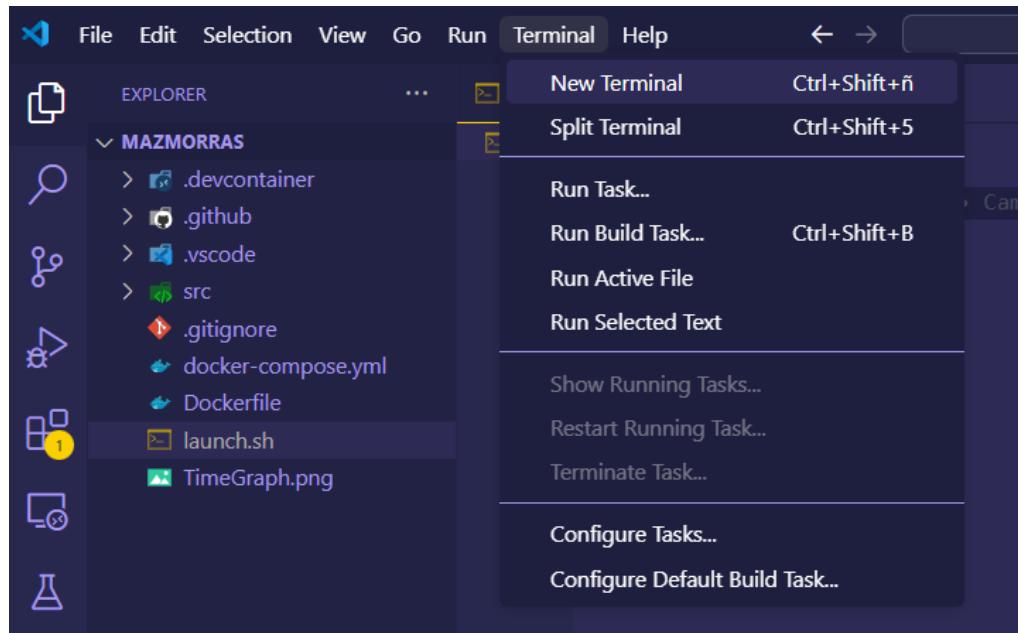


Figura D.10: Barra de opciones de Visual Studio Code

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección se va a desarrollar el uso que se puede hacer de la aplicación por parte del usuario. Este proyecto se compone de dos partes, un ejecutable y el servidor.

E.2. Requisitos de usuarios

Este proyecto en el estado en el que se encuentra ahora se necesita arrancar el servidor en la máquina localmente pero en líneas futuras, esto no sería necesario para el usuario ya que está preparado para ser alojado en un servidor externo.

A continuación se van a presentar los requisitos que se necesita por parte del usuario:

- **Sistema operativo:** Debe de tener un sistema operativo Windows 10 o superior o Linux Ubuntu 18.04 o superior
- **Procesador:** Intel Core i3 o equivalente
- **Memoria RAM:** 4GB
- **Espacio en el disco:** Al menos 64GB de espacio libre

E.3. Instalación

Para poder ejecutar el proyecto en el dispositivo, se necesita instalar el Docker para poder arrancar el servidor en local. El proceso de instalación es igual a lo mencionado anteriormente en el manual del programador. Se necesita descargar el repositorio de la siguiente url <https://github.com/etc99/algoritmosYmazmorras>.

E.4. Manual del usuario

Manual del servidor

Para poder visualizar los laberintos en el juego, primero es necesario tener arrancado el servidor. Para realizarlo es necesario abrir la terminal.

En la terminal, con la ruta del directorio apuntando a /Mazmorras se necesita ejecutar el siguiente comando para levantar el servidor:

```
1 docker compose up
```

Para poder parar el contenedor del juego, desde el teclado se pulsarían los botones Ctrl+C y para borrar el contenedor se realiza con el siguiente comando:

```
1 docker compose down
```

Manual del juego

Teniendo ya listo el servidor, ahora solo queda lanzar el ejecutable del videojuego. En la carpeta del ejecutable aparecerán diversos archivos, pero el ejecutable es el que se encuentra marcado en la figura E.1.

📁	AlgoritmosYMazmorras_Data	04/07/2024 17:59	Carpeta de archivos
📁	MonoBleedingEdge	04/07/2024 17:59	Carpeta de archivos
⚙️	AlgoritmosYMazmorras	04/07/2024 17:59	Aplicación 639 KB
⚙️	UnityCrashHandler64	04/07/2024 17:59	Aplicación 1.098 KB
📄	UnityPlayer.dll	04/07/2024 17:59	Extensión de la ap... 28.686 KB

Figura E.1: Carpeta del ejecutable de Unity.

Tras esto la primera pantalla que se mostrará es el menú principal como se muestra en la figura E.2. Para poder avanzar a la siguiente pantalla se necesita pulsar sobre el botón de «¡Comenzar!», al hacerlo se muestra la pantalla de selección de algoritmo E.3.

Tras seleccionar uno de los algoritmos en la siguiente se mostrará lo que va a ser el tablero y el jugador, y en el margen izquierdo aparecerá un menú en el que se podrán introducir los parámetros deseados para el laberinto E.4.

Después de introducir los parámetros se pulsa sobre el botón de «Generar». Tras realizar esto se muestra el resultado para poder navegar el laberinto con las teclas «W, A, S, D» E.5.

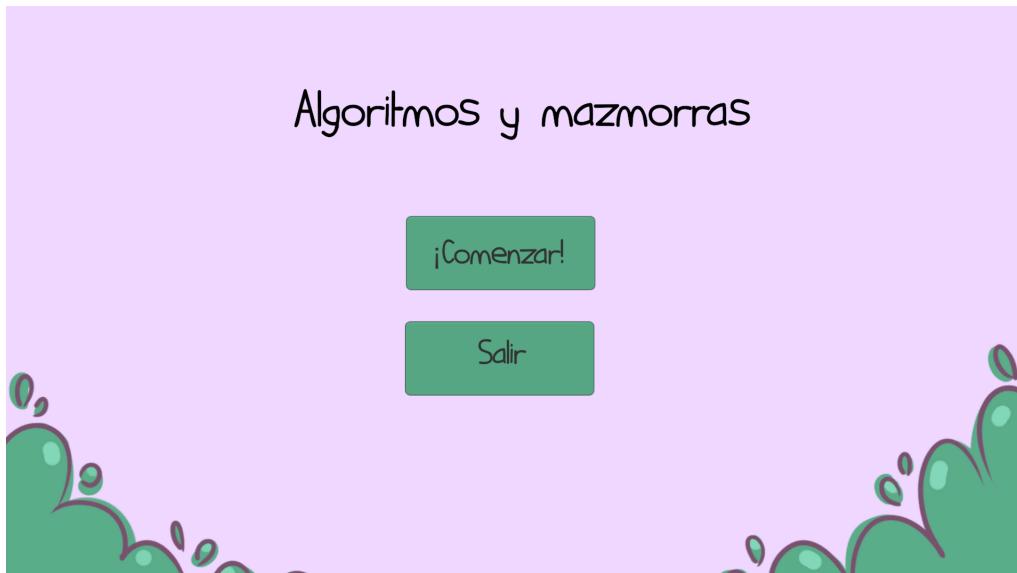


Figura E.2: Captura del menú principal del juego.



Figura E.3: Captura del menú de selección de algoritmo.

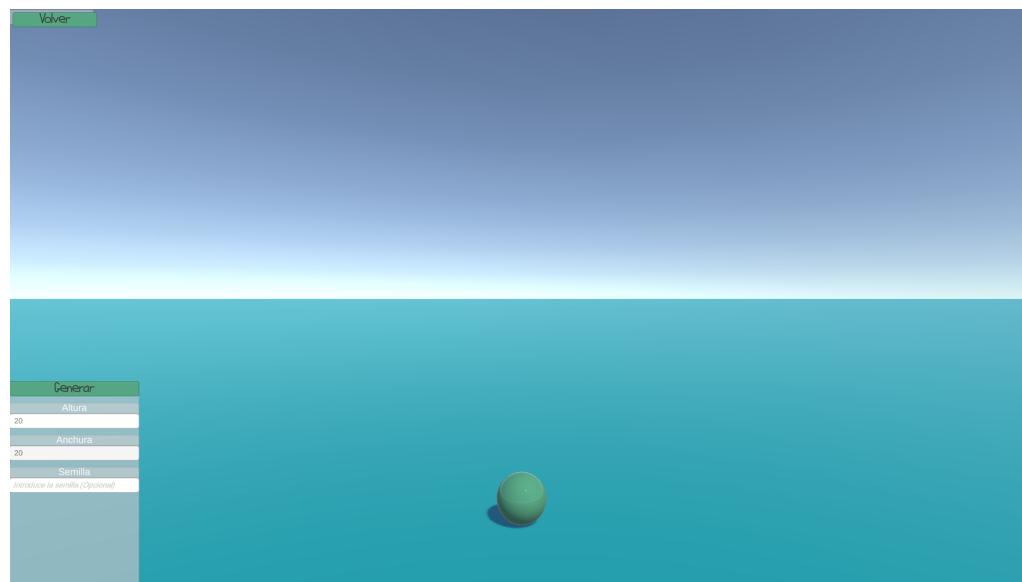


Figura E.4: Captura de la interfaz para introducir los parámetros del laberinto.

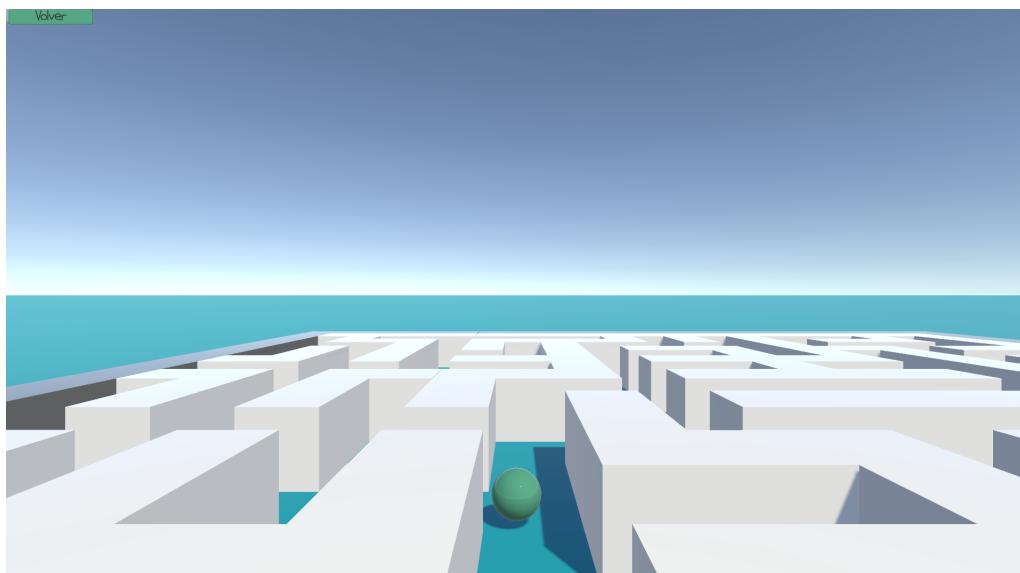


Figura E.5: Captura del laberinto generado.

Apéndice F

Sostenibilización curricular

F.1. Introducción

El documento “Directrices para la introducción de la Sostenibilidad en el Curriculum” aprobado por la CRUE establece una serie de objetivos clave para la integración de la sostenibilidad en el currículo de las universidades españolas. Estos objetivos tienen como propósito fundamental fomentar un desarrollo humano sostenible a través de la educación superior. A continuación, se detallan los principales objetivos planteados:

F.2. Integración Transversal de la Sostenibilidad

Objetivo: Incluir la sostenibilidad como un eje transversal en todas las titulaciones universitarias, asegurando que todos los estudiantes adquieran competencias en este ámbito, independientemente de su área de estudio.

Acciones: Se promoverá la revisión y actualización de los planes de estudio para incluir contenidos específicos y transversales sobre sostenibilidad, adaptados a cada disciplina.

F.3. Desarrollo de Competencias para la Sostenibilidad

Objetivo: Desarrollar en los estudiantes las competencias necesarias para identificar y resolver problemas socioambientales desde una perspectiva

sistémica y ética.

Acciones: Los planes de estudio deberán contemplar la formación en competencias como el pensamiento crítico, la gestión sostenible de recursos, la participación en procesos comunitarios y la aplicación de principios éticos en la vida personal y profesional.

F.4. Fomento de la Investigación y Docencia Sostenible

Objetivo: Promover la investigación y la docencia que integren los principios del desarrollo sostenible, asegurando que la actividad académica contribuya a la solución de problemas ambientales y sociales.

Acciones: Se incentivará la realización de proyectos de investigación y la impartición de cursos que aborden la sostenibilidad desde diversas perspectivas, facilitando la formación continua y el aprendizaje a lo largo de la vida.

F.5. Sensibilización y Participación de la Comunidad Universitaria

Objetivo: Involucrar a toda la comunidad universitaria en el compromiso con la sostenibilidad, fomentando la participación activa en la implementación de políticas y acciones sostenibles dentro y fuera del campus.

Acciones: Se promoverán actividades extracurriculares, seminarios, talleres y jornadas que sensibilicen a estudiantes, profesores y personal administrativo sobre la importancia de la sostenibilidad y su aplicación en la vida diaria.

F.6. Establecimiento de Mecanismos de Evaluación y Mejora Continua

Objetivo: Implementar sistemas de evaluación que aseguren la calidad y efectividad de las acciones educativas en sostenibilidad, permitiendo una mejora continua de los programas académicos y las prácticas institucionales.

Acciones: Se incluirán criterios de sostenibilidad en los sistemas de evaluación de la calidad universitaria y en la evaluación del profesorado, ga-

rantizando que tanto la docencia como la investigación se alineen con los principios del desarrollo sostenible.

F.7. Creación de Redes y Plataformas de Colaboración

Objetivo: Establecer redes de colaboración entre universidades y otras entidades para intercambiar experiencias y buenas prácticas en la integración de la sostenibilidad en el currículo académico.

Acciones: Se fomentará la creación de grupos de trabajo interuniversitarios y la participación en plataformas estatales e internacionales que faciliten el intercambio de conocimientos y la cooperación en proyectos de sostenibilidad.

Estos objetivos buscan no solo la formación de profesionales capacitados para enfrentar los desafíos actuales y futuros, sino también la creación de una cultura universitaria comprometida con la sostenibilidad y el desarrollo humano integral. La implementación efectiva de estas directrices requiere del compromiso institucional y la participación activa de toda la comunidad universitaria.^[2]

Bibliografía

- [1] Apache. Apache license, version 2.0. <https://www.apache.org/licenses/LICENSE-2.0.html>, 2024. Accessed: 2024-06-16.
- [2] CRUE Universidades Españolas. Directrices para la sostenibilidad ambiental de la universidad en el siglo xxi. https://www.crue.org/wp-content/uploads/2020/02/Directrices_Sostenibilidad_Crue2012.pdf, 2020. Accessed: 2024-06-16.
- [3] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, 2012. Can be accessed online via [Google Books](#).
- [4] Refactoring Guru. Patrón de diseño estrategia. <https://refactoring.guru/es/design-patterns/strategy>, 2024. Accessed: 2024-06-16.
- [5] Refactoring Guru. Patrón de diseño método plantilla. <https://refactoring.guru/es/design-patterns/template-method>, 2024.
- [6] Open Source Initiative. The 3-clause bsd license. <https://opensource.org/license/BSD-3-clause>, 2024. Accessed: 2024-06-16.
- [7] Jobted. Sueldo del programador en españa. <https://www.jobted.es/salario/programador>, 2024. Accessed: 2024-06-16.
- [8] KeepCoding. ¿qué es el principio de sustitución de liskov? <https://keepcoding.io/blog/que-es-el-principio-de-sustitucion-de-liskov/>, 2024.
- [9] Microsoft. Buy and download windows 11 pro. <https://www.microsoft.com/en-us/d/windows-11-pro/dg7gmgf0d8h4>, 2024. Accessed: 2024-06-16.

- [10] Microsoft. Uso de contenedores de docker en wsl 2. <https://learn.microsoft.com/es-es/windows/wsl/tutorials/wsl-containers>, 2024. Accessed: 2024-06-16.
- [11] MSI. Overview raider ge66 12uh. <https://www.msi.com/Laptop/Raider-GE66-12UH>, 2024. Accessed: 2024-06-16.
- [12] TRBL Services. The open/closed principle. <https://trbl-services.eu/blog-solid-open-closed/>, 2024.
- [13] Seguridad Social. Cotización y recaudación de trabajadores. <https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537>, 2024.
- [14] Unity Technologies. Unity documentation. <https://docs.unity3d.com/>, 2024.