



UNIVERSIDAD DE MÁLAGA



Graduado en Ingeniería del Software

Diseño y desarrollo de herramienta interactiva y modelos subrogados para la optimización de flujos de movimientos de personas en situaciones de emergencias en escenarios cerrados

Design and development of an interactive tool and surrogate models for the optimisation of people movement flows in emergency situations in closed scenarios

Realizado por
Ángel Fernández Civila

Tutorizado por
Carlos Cotta Porras
Antonio José Fernández Leiva

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre de 2024



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

**Diseño y desarrollo de herramienta interactiva y modelos
subrogados para la optimización de flujos de
movimientos de personas en situaciones de emergencias
en escenarios cerrados**

**Design and development of an interactive tool and
surrogate models for the optimisation of people
movement flows in emergency situations in closed
scenarios**

Realizado por
Ángel Fernández Civilá

Tutorizado por
Carlos Cotta Porras
Antonio José Fernández Leiva

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2024

Fecha defensa: septiembre de 2024

Abstract

The work proposed in this project is framed within the lines of work of the national research project entitled “New bio-inspired optimisation techniques for resilience and sustainability (*Bio4Res*)” with reference PID2021-125184NB-I00 carried out by the CAESIUM research group [CAESIUM2023] [17] from the ITIS UMA (Instituto de Tecnologías e Ingeniería del Software, Universidad de Málaga).

In the Bio4Res project [16], bio-inspired optimisation methods (BOMs) and resilience are researched. On the one hand, this involves a study of the elements and methods by which these techniques can be made resilient to different types of disruptions, as well as the conditions under which these techniques can be executed in a sustainable and environmentally friendly way. On the other hand, the application of BOMs to domains where resilience is wanted is being considered. It is precisely at this point where this final degree project is framed, whose objective is twofold:

- On the one hand, to facilitate the configuration of environments/scenarios in which the BOMs resulting from the Bio4Res project can be applied and allow the results obtained to be graphically visualised. The objective is to adopt an optimisation approach based on simulations, so that a controlled scaling of the different factors that complicate the resolution of these problems can be carried out. The main objective has been the development of a tool that allows the configuration and visualisation of the simulation of real scenarios in which movements of people are simulated in emergency scenarios that affect the citizens.
- On the other hand, one of the main problems of the BOMs generated in the Bio4Res project to optimise the evacuation of people in distress in indoor scenarios when an emergency situation arises is their high computational cost, and in order to reduce this computational cost, this project will consider the design and development of a surrogate model generated through machine learning that replaces the calculation of quality, i.e. fitness, in the

BOMs used.

Keywords: Bio-inspired optimisation, resilience, simulation, surrogate model, machine learning

Resumen

El trabajo cuyo desarrollo se plantea en este proyecto está enmarcado dentro de las líneas de trabajo del proyecto de investigación nacional titulado “Nuevas técnicas de optimización bioinspirada para resiliencia y sostenibilidad (*Bio4Res*)” con referencia PID2021-125184NB-I00 llevado a cabo por el grupo de investigación CAESIUM [CAESIUM2023] [17] de la UMA desde el Instituto de Tecnología e Ingeniería del Software (ITIS).

En el proyecto Bio4Res [16] se investiga en métodos de optimización bioinspirada (MOBs) y resiliencia. Por un lado, esto supone un estudio de los factores y métodos a través de los cuales estas técnicas pueden ser dotadas de resiliencia a diferentes tipos de disrupciones, así como las condiciones bajo las que estas técnicas pueden ejecutarse de manera sostenible y respetuosa con el medio ambiente. Por otro lado, se está considerando la aplicación de MOBs a dominios en los que se busca resiliencia. Es precisamente en este punto donde se enmarca este TFG cuyo objetivo es doble:

- Por una parte, facilitar la configuración de entornos/escenarios sobre los cuales poder aplicar los MOBs resultantes del proyecto Bio4Res y permitir visualizar gráficamente los resultados obtenidos. Se trata de adoptar un enfoque de optimización basado en simulaciones, de manera que se puede realizar un escalado controlado de los diferentes factores que complican la resolución de estos problemas. El objetivo principal ha sido el desarrollo de una herramienta que permite configurar y visualizar la simulación de escenarios reales en los cuales se simulan movimientos de personas en escenarios de emergencia que afectan a la ciudadanía.
- Por otra parte, uno de los problemas principales de los MOBs generados en el proyecto Bio4Res para optimizar la evacuación de personas aterrorizadas en escenarios cerrados en los cuales surge alguna situación de emergencia es su elevado coste computacional, y con el fin de reducir este coste computacional, este proyecto va a considerar el diseño y desarrollo de un modelo

subrogado generado a través de técnicas de aprendizaje automático que sustituya al cálculo de la calidad, es decir, el fitness, en los MOBs empleados.

Palabras clave: Optimización bioinspirada, resiliencia, simulación, modelo subrogado, técnicas de aprendizaje automático

Índice

1. Introducción	15
1.1. Motivación	15
1.2. Objetivos	17
1.3. Metodología de trabajo	17
1.4. Estructura del documento	18
2. Antecedentes	19
2.1. Tecnologías	19
2.1.1. Unity	19
2.1.2. Paquetes de Unity	20
2.1.3. C#	20
2.1.4. Rider, Visual Studio o Visual Studio Code	21
2.1.5. Java	22
2.1.6. IntelliJ IDEA	22
2.1.7. JSON	22
2.1.8. Git y Github	22
2.1.9. Python	23
2.1.10. PyCharm o Jupyter Lab o Visual Studio Code	23
2.1.11. Paquetes de Python necesarios	24
2.1.12. R y RStudio	25
2.1.13. LaTeX y Overleaf	25
2.2. Técnicas de IA	25
2.2.1. Autómata celular	26
2.2.2. Algoritmo evolutivo	28
2.2.3. Machine learning	29
3. Diseño	31
3.1. Actores del sistema	31

3.2.	Requisitos funcionales y no funcionales	31
3.2.1.	Requisitos funcionales	31
3.2.2.	Requisitos no funcionales	32
3.3.	Casos de uso	33
3.3.1.	CU 1: Modificación de los parámetros para las simulaciones.	33
3.3.2.	CU 2: Iniciación de una simulación.	34
3.3.3.	CU 3: Generación de un escenario aleatorio.	37
3.3.4.	CU 4: Modificación de la velocidad de la simulación.	38
3.3.5.	CU 5: Alternar entre las dos vistas posibles.	39
3.3.6.	CU 6: Cargar una traza externa.	40
3.3.7.	CU 7: Cargar un escenario externo.	42
3.3.8.	CU 8: Guardar la traza que se ejecutará en la simulación.	43
3.3.9.	CU 9: Guardar el escenario que se usará en la simulación.	45
3.4.	Diseño del modelo surrogado	46
4.	Implementación	49
4.1.	Arquitectura de la herramienta	49
4.2.	Diagrama de clases	50
5.	Pruebas	63
5.1.	Pruebas de usabilidad	64
6.	Resultados	65
6.1.	Modificar parámetros	65
6.2.	Cargar escenario	66
6.3.	Guardar escenario	67
6.4.	Cargar traza	68
6.5.	Guardar traza	69
6.6.	Generar un escenario aleatorio	71
6.7.	Iniciar simulación	71
6.8.	Alternar la vista	72
6.8.1.	Controlar la cámara	72

6.9. Alterar la velocidad de simulación	73
7. Experimentación	75
8. Conclusiones y Líneas Futuras	81
8.1. Conclusiones	81
8.2. Líneas Futuras	82
Apéndice A. Manual de Instalación	89

Índice de figuras

1.	Vecindad de Von Neumann	27
2.	Vecindad de Moore	27
3.	Proceso del algoritmo evolutivo del proyecto	29
4.	Capas del modelo implementads utilizando Keras	47
5.	Estructura del Autómata celular	51
6.	Estructura del escenario	52
7.	Estructura del Floor Field	53
8.	Estructura de las casillas vecinas	53
9.	Estructura de los agentes	54
10.	Clase para cargar y guardar los JSON	55
11.	Estructura para los archivos JSON de las trazas	56
12.	Estructura para los archivos JSON de los escenarios	56
13.	Tipos de formas en el Json del escenario para los obsáculos y puertas	57
14.	Controlador para la UI y las rutas para los JSON	58
15.	Controlodares para la escena en Unity	59
16.	Tipos de eventos de la herramienta	60
17.	Clase para las estadísticas de una simulación	61
18.	Campos disponibles	65
19.	Cargar escenario activado	66
20.	Explorador de archivos para elegir el escenario	66
21.	Ruta de escenario guardada	67
22.	Guardar escenario activado	67
23.	Explorador de archivos para elegir la ruta para el escenario	68
24.	Ruta para escenario guardada	68
25.	Cargar traza activado	68
26.	Explorador de archivos para elegir la traza	69
27.	Ruta de traza guardada	69
28.	Guardar traza activado	70

29.	Explorador de archivos para elegir la ruta para la traza	70
30.	Ruta para traza guardada	71
31.	Botón para generar un escenario aleatorio	71
32.	Botón para iniciar una simulación	71
33.	Casilla para alternar la vista	72
34.	Vista desde arriba	72
35.	Vista libre	73
36.	Barra para modificar la velocidad de las simulaciones	73
37.	Escenario base	77
38.	Los mejores fitness de cada algoritmo	78
39.	Los fitness de las mejores soluciones de cada algoritmo	78
40.	Tabla con varias estadísticas de cada algoritmo en cada instancia	79
41.	La mejor solución para cada algoritmo en cada instancia	79

Índice de cuadros

1.	CU 1: Modificación de los parámetros para las simulaciones.	33
2.	CU 2: Iniciación de una simulación.	34
3.	CU 3: Generación de un escenario aleatorio.	37
4.	CU 4: Modificación de la velocidad de la simulación.	38
5.	CU 5: Alternar entre las dos vistas posibles.	39
6.	CU 6: Cargar una traza externa.	40
7.	CU 7: Cargar un escenario externo.	42
8.	CU 8: Guardar la traza que se ejecutará en la simulación.	43
9.	CU 9: Guardar el escenario que se usará en la simulación.	45

1

Introducción

Este capítulo introduce lo que es este proyecto a rasgos generales.

1.1. Motivación

Un buen diseño de un entorno cerrado en el cual fluctúan personas (entrando y saliendo) puede facilitar la evacuación de estas cuando surge una situación de emergencia que pone en peligro la integridad de las mismas. Así, la inteligencia artificial, junto a los expertos humanos (por ejemplo arquitectos o diseñadores de interiores) ya ha sido aplicada para la optimización del diseño de espacios con este fin, o sea, el de permitir la rápida evacuación de las personas en espacios cerrados. Este proyecto se contextualiza en este contexto, abordando el problema desde la perspectiva de optimizar la colocación de las puertas de entrada y salida en espacios cerrados en los cuales hay un movimiento dinámico y constante de personas que entran y salen [6]. La colocación óptima de esas puertas se dirige a facilitar la evacuación de las personas o peatones que están dentro del espacio cerrado ante situaciones de emergencia. De manera muy resumida:

1. Para simular el comportamiento de los peatones en esos entornos o escenarios con una distribución de salidas específica teniendo en cuenta el propio entorno, los peatones y las interacciones entre ellos, se ha utilizado un autómata celular.
2. A continuación, se propone una métrica para determinar la calidad de la solución de una evacuación. Esta métrica es conocida como *fitness*.
3. Una vez esto, se ha propuesto un algoritmo metaheurístico llamado algoritmo evolutivo (EA) para determinar el éxito de la evacuación o la calidad de la solución (*fitness*) ejecutando el autómata celular repetidas veces.

4. Se ha comprobado que el algoritmo evolutivo propuesto es capaz de hallar soluciones eficaces para distintos escenarios.

A partir de esta solución surgen dos necesidades: poder configurar y observar una evacuación en cualquier escenario ya que es poco configurable la visualización desde ese proyecto, e intentar reducir el tiempo de ejecución puesto que la ejecución repetida del autómatas celular tiene un alto coste computacional.

Gracias a estas dos necesidades, se plantea el inicio de este proyecto para el TFG.

- Para la primera necesidad, desarrollar una herramienta para poder configurar y visualizar evaluaciones en cualquier entorno que sea externa a la solución anterior. Esta herramienta se ha desarrollado utilizando un motor de videojuegos puesto que estos están optimizados para emulaciones 2D y 3D en tiempo real. La herramienta deberá permitir generar evacuaciones nuevas usando el autómatas celular y visualizar evacuaciones en escenarios ya existentes. Para esto último, es necesario extraer los datos de las evacuaciones y de los escenarios para poder visualizar la evacuación desde la herramienta. Como además de lo mencionado, en la herramienta también se pueden configurar algunos parámetros del autómatas celular, se considera un éxito.
- En el caso de la segunda necesidad, aunque a partir del algoritmo evolutivo se obtienen soluciones efectivas, este tiene un coste computacional excesivo, necesitando una gran cantidad de tiempo para poder hallar esas soluciones. El problema está en que para calcular la calidad de la solución, es necesario usar el autómatas celular, y ejecutarlo repetidas veces es mucho coste computacional. Para evitar que se tarde una cantidad de tiempo excesiva, se ha planteado sustituir el autómatas celular del que se obtiene el *fitness* por un modelo subrogado que lo calcule directamente. Este modelo se entrenará utilizando *machine learning* y debería recibir el conjunto de salidas que recibe el autómatas celular dentro del EA y devolver un *fitness* no muy alejado del que se calcularía utilizando el autómatas celular, aunque lo que se busca es demostrar que la sustitución es viable de cara al futuro. A pesar de que las calidades de las soluciones que se calculan usando el modelo son muy dispares de las obtenidas usando el autómatas celular, se ha podido sustituir el autómatas celular por el modelo subrogado y funciona, por lo que no se consideraría un fracaso, aunque tampoco un éxito.

1.2. Objetivos

El objetivo principal de este trabajo de fin de grado es el desarrollo de una herramienta que permita configurar y visualizar la simulación de escenarios reales en los cuales se simulen movimientos de personas mediante un autómata celular en escenarios de emergencia que afecten a la ciudadanía. La herramienta a crear en el seno de este TFG se empleará dentro del proyecto de investigación Bio4Res.

Otro objetivo a realizar es un algoritmo que distribuya las salidas en uno de los escenarios anteriores de la manera más eficiente para que los ciudadanos/as puedan salir ilesos. Este algoritmo ya está implementado por el grupo de investigación en Java mediante un algoritmo evolutivo. Sin embargo, debido a que para completarlo se necesita ejecutar el autómata celular numerosas veces, el objetivo se ha convertido en intentar optimizar el tiempo que dura este algoritmo evolutivo usando un modelo de *machine learning* en lugar del autómata celular.

1.3. Metodología de trabajo

Para el desarrollo de esta herramienta, se ha seguido la metodología de desarrollo ágil de software Scrum [23], la cual es una de las más usadas. Esta metodología es un procedimiento que consiste en llevar a cabo un conjunto de tareas de forma regular y constante con el objetivo de trabajar eficientemente y de manera colaborativa. Para trabajar de forma regular, se han repartido las tareas en distintos periodos de tiempo a lo largo del proyecto, llamando sprints a estos ciclos de trabajo. Al comienzo del proyecto, se ha creado un backlog del producto, el cual es una lista con todas las tareas a realizar del proyecto ordenada según la prioridad de estas. Antes de empezar un sprint, era necesario planificar cada sprint, asignando que tareas del backlog están pensadas para finalizarlas en ese sprint, y para verificar que se ha cumplido con las expectativas o en caso de que no, adaptarse a las circunstancias, al final de cada sprint se planificó una breve reunión para tratarlo.

En el caso de la obtención de requisitos, como el grupo de investigación ya tenía una idea de como podría funcionar la herramienta de simulación, el tutor de este TFG hizo de cliente, aportando los requisitos necesarios del proyecto que se fueron implementando a lo largo de los sprints, que finalizaban con una reunión con él en donde se resumía como había avanzado el proyecto en ese ciclo de trabajo.

Tras la primera etapa de análisis y definición de requisitos iniciales, se llevó a cabo el diseño de la herramienta para empezar posteriormente con la implementación en el motor Unity según las tareas que se habían asignado en el sprint actual.

También se ha escrito una memoria de cada fase implementada, que se ha ido actualizando con cada fase del proyecto conforme se vayan terminando.

1.4. Estructura del documento

Esta memoria se encuentra compuesta por los siguientes apartados junto a una breve descripción:

1. **Introducción:** Donde se encuentra descrita la motivación, los objetivos y la estructura de la memoria junto a una descripción breve para cada apartado.
2. **Antecedentes:** Apartado en donde se menciona y explica los conceptos que son relevantes en este proyecto pero que no sean novedosos en sus respectivos campos.
3. **Diseño:** Donde se definen los requisitos (tanto los funcionales como los no funcionales) y los casos de usos.
4. **Implementación:** Describe la arquitectura de la aplicación de simulación y del modelo de machine learning, además de explorar los diagramas de clase de la herramienta.
5. **Pruebas:** Explicación sobre las pruebas realizadas a lo largo del proyecto.
6. **Resultados:** Descripción de los pasos tomados para la implementación del proyecto.
7. **Experimentación:** Comentarios y detalles sobre los tests estadísticos ejecutados para comparar el EA y el modelo surrogado.
8. **Conclusiones y Líneas Futuras:** En este último apartado se valora el proyecto, y así poder obtener conclusiones sobre los problemas encontrados a lo largo de este, además de pensar en mejoras que puedan llegar a ser viables a futuro para añadirlas en el proyecto.

2

Antecedentes

Este apartado sirve para mencionar y explicar los conceptos que son relevantes en este proyecto pero que no sean novedosos en sus respectivos campos, como las tecnologías usadas abarcando desde herramientas hasta lenguajes de programación o librerías, o las técnicas de IA usadas.

2.1. Tecnologías

Para la realización de este proyecto, se han usado distintas tecnologías de acuerdo a los dos objetivos que se perseguían. Para el primer objetivo (la herramienta de simulación) se ha usado el motor de videojuegos Unity junto al lenguaje de programación C# y un paquete externo para los JSON, siendo estas tecnologías descritas brevemente desde la sección 2.1.1 hasta la 2.1.8. En cuanto al objetivo del algoritmo que distribuye optimamente las salidas, se ha utilizado Java junto a varias librerías de Bio4Res ¹, además de Python y varias librerías, que se describen desde el apartado 2.1.5 hasta el 2.1.11.

2.1.1. Unity

Unity [26] es un motor de videojuegos que salió en 2005, el cual es el más popular debido entre otras razones a su versatilidad y amplia comunidad. Este permite realizar proyectos en 2D, 3D, realidad aumentada (AR), realidad virtual (VR) e incluso realidad extendida (XR). El uso de Unity para este proyecto está justificado por los siguientes motivos:

- Es uno de los motores gráficos de videojuegos más veteranos, estando bastante pulido y teniendo una de las comunidades de desarrolladores más grandes.

¹[Espacio de Bio4Res en Github](#)

- Es de las herramientas más fáciles de aprender por su intuitiva interfaz y conceptos, junto a una extensa documentación, y siendo C# fácil de aprender también. Además, al tener una comunidad grande, hay muchas cuestiones resueltas en foros.
- Es gratuito para el uso de proyectos personales y proyectos que no supere cierta cantidad de dinero ganado con el proyecto.

2.1.2. Paquetes de Unity

Una de las razones que se han mencionado de para que se haya optado por Unity para desarrollar la herramienta de visualización de simulaciones es que su comunidad es una de las más extendidas. Debido a esto, existen varios paquetes para Unity, de los cuales he usado los siguientes

- *UnitySimpleFileBrowser* [8]: Para este proyecto, era necesario buscar archivos y rutas del ordenador, y la mejor manera para hacerlo era usando un explorador de archivos. Sin embargo, había dos prioridades: tenía que ser simple y minimalista, y debía funcionar en todas los sistemas operativos, por lo que no valía el que tiene Windows por defecto. Y buscando, encontré este framework que permite abrir y personalizar la configuración de un explorador de archivos simple.
- *Newtonsoft Json* [2]: Una funcionalidad necesaria de esta herramienta era cargar, guardar y usar los JSON². Unity tiene integrado un paquete que permite todo esto. Sin embargo, puntualmente tiene problemas, sobretudo a la hora de leer y escribir números decimales en el JSON, cambiando las décimas. Como se requería de la mayor precisión posible, se ha decidido descartar este paquete y usar *Newtonsoft Json*, que es un framework JSON de alto rendimiento para .NET. Este no solo maneja los datos del JSON correctamente, sino que también optimiza ligeramente el proceso de lectura y guardado de estos datos.

2.1.3. C#

C# [24] es un lenguaje de programación multiparadigma de alto nivel desarrollado por Microsoft para .NET, el cual es usado en varias aplicaciones gracias a su versatilidad y su capacidad de desarrollo. Este lenguaje es uno de los diseñados para la infraestructura de lenguaje

²JavaScript Object Notation

común. Sus usos más frecuentes es en el desarrollo de aplicaciones de escritorio, de móvil y web, en el desarrollo de videojuegos (con Unity), desarrollo de microservicios, y desarrollo de librerías y componentes para .NET además de varios más.

2.1.4. Rider, Visual Studio o Visual Studio Code

Para poder escribir y compilar código en cualquier lenguaje, lo mínimo que se necesita es un editor de texto para escribirlo y un compilador para verificar que no hay errores. Sin embargo, a día de hoy, se suele utilizar un IDE³. Los IDE además de empaquetar esas dos funcionalidades en una sola herramienta, también proporcionan otras utilidades dependiendo del IDE para facilitar a los desarrolladores. Para este proyecto, era necesario un IDE que pudiese compilar C# y que prestase facilidades para poder integrarlo con Unity. Aunque cualquiera de las tres opciones son válidas, cada una tiene sus ventajas e inconvenientes, y en mi caso la que utilicé fue Rider.

- *Visual Studio Code* [19] es gratuito, ligero y fácil de usar. Funciona mediante extensiones que realiza su comunidad. Esto conlleva a que dependiendo del uso que se le quiera dar, es necesario instalarse unas extensiones específicas.
- *Visual Studio* [18] también es gratuito y dependiendo de los módulos que se instale, se puede enfocar en varios lenguajes de programación como C#, C++, .NET, etc. También tiene varias utilidades que pueden aportar al desarrollador, aunque esto puede conllevar a que el programa no sea del todo intuitivo de usar.
- *Rider* [12] es un IDE que está enfocado para el desarrollo de juegos y para .NET, por lo que a diferencia de los dos IDE anteriores, este ofrece la integración con el motor de videojuegos sin necesidad de instalar ningún módulo o extensión. Sin embargo, este no es gratuito y requiere de una tarifa mensual o anual para poder usarlo. No obstante JetBrains ofrece acceso a varias herramientas suyas por ser integrante de la UMA, entre las que se encuentra Rider.

³Entorno de desarrollo integrado

2.1.5. Java

Java [7] es uno de los lenguajes de programación multiparadigma de alto nivel más conocidos y usados. Lleva siendo utilizado en la industria del software desde hace bastante tiempo, y a día de hoy se sigue utilizando debido a su portabilidad, versatilidad y robustez. Es usado principalmente para Aplicaciones Empresariales (Java EE), aplicaciones móviles y de escritorio, sistemas empujados, sistemas de control y automatización, y automatización de pruebas entre muchos otros usos.

2.1.6. IntelliJ IDEA

Al igual que es necesario un IDE para C#, también lo es para Java y Python. En el caso de Java, el más conocido y actualizado es *IntelliJ IDEA* [14], por lo que para este proyecto se ha usado este IDE ya que además de facilitar la integración de los proyectos Maven, siendo necesario importar como librerías varios proyectos realizados por el proyecto Bio4Res, es el IDE con el que se está más familiarizado en el grupo de investigación.

2.1.7. JSON

Para este proyecto es necesario almacenar información y poder extraerla. Debido a esto y a que es uno de los formatos más usados, se ha escogido el formato de texto *JSON* [20], cuya sintaxis deriva de la de JavaScript ya que forma parte de este. En este proyecto los JSON cumplen un papel importante, ya que se encargan de almacenar la información tanto de entrada como de salida del algoritmo evolutivo, pudiendo utilizar esa información para otros usos, como los tests estadísticos o el entrenamiento de los modelos subrogados.

2.1.8. Git y Github

A pesar de que este proyecto es realizado por una persona, debido a su tamaño y constantes cambios, se ha considerado necesario el uso de un controlador de versiones, siendo *Git* [3] el escogido por su gran popularidad y por ser el software de control de versiones con el que el grupo de investigación estaba familiarizado. Esto permite ver y manejar cada versión del proyecto que se haya subido al repositorio correspondiente, además de poder ver los cambios que se han realizado entre cada versión.

Por las mismas razones que Git, se ha escogido *Github* [3] como plataforma de gestión de repositorios de código. Esta plataforma sirve para almacenar y gestionar el código fuente de sus proyectos, facilitando varias tareas de Git. Esta plataforma permite ver el código de cada versión y sus cambios entre estas de una forma más clara y sencilla que desde Git, además de descargar el código de cualquier versión que se haya subido al repositorio. Github también permite añadir a otros colaboradores al repositorio para que puedan manejar el proyecto y que sea posible tener el mismo proyecto en varias máquinas.

2.1.9. Python

Python es un lenguaje de programación multiparadigma, también de alto nivel, conocido por su versatilidad y por su legibilidad. Estas razones lo hacen a día de hoy uno de los lenguajes más populares y usados en varios ámbitos de la informática, como la ciberseguridad, servidores webs y conexión de bases de datos, estadística, inteligencia artificial y machine learning, siendo este último el motivo principal por el que se ha decidido utilizar para este proyecto.

2.1.10. PyCharm o Jupyter Lab o Visual Studio Code

Como se ha mencionado en el apartado 2.1.6, también es necesario un IDE para Python. Entre los IDE más conocidos está PyCharm [11], y se ha escogido entre los demás ya que facilita la creación y el uso de un entorno virtual de Python junto a Git y Github, y al igual que IntelliJ IDEA, es de JetBrains, por lo que ambos programas comparten varias similitudes. Aun así, hay otras opciones muy buenas para poder desarrollar en Python:

- *PyCharm* es un IDE que está enfocado para el desarrollo de juegos y para .NET, por lo que a diferencia de los dos IDE anteriores, este ofrece la integración con el motor de videojuegos sin necesidad de instalar ningún módulo o extensión. Sin embargo, este no es gratuito y requiere de una tarifa mensual o anual para poder usarlo.
- *Jupyter Lab* [31] también es gratuito, y a diferencia de los demás IDE mencionados, se ejecuta en la web y permite crear documentos llamados “Jupyter notebook”, que se pueden convertir a varios formatos como HTML, LaTeX, Markdown o Python entre otros. Estos documentos suelen terminar con la extensión “.ipynb”, y tienen como ventaja que

se pueden escribir en ellos texto o importar imágenes entre otros elementos al mismo tiempo que escribir, compilar y ejecutar el código del documento.

- Como se ha explicado anteriormente, *Visual Studio Code* tiene varias extensiones, por lo que dependiendo de las que se instalen este tendrá un uso específico.

2.1.11. Paquetes de Python necesarios

Para la parte de aprendizaje automático del TFG, es necesario importar algunos módulos para Python que serán fundamentales para esta labor:

- *NumPy* [29] es una biblioteca de código abierto que permite crear y manejar fácilmente vectores y matrices multidimensionales mediante el uso de su propia estructura de datos “ndarray”. Estos funcionan de forma distinta a las listas de Python, siendo la estructura “ndarray” más eficiente y teniendo más compatibilidad con las bibliotecas numéricas actuales. Sin embargo, debido a que su funcionamiento es distinto, tienen algunas limitaciones, como que al concatenar dos matrices no se vinculan realmente, sino que devuelve una nueva.
- *Matplotlib* [10] es una de las bibliotecas de código abierto más populares de generación de gráficas en dos dimensiones y visualización de datos, ya sean estas visualizaciones animadas, estáticas o interactivas. Necesitamos esta biblioteca para poder usar el módulo “pyplot”, que nos permitirá crear gráficos lineales en dos dimensiones.
- *TensorFlow* [25] es de las bibliotecas más conocidas de modelos de aprendizaje automático. Esta es una biblioteca de código abierto creada por Google para sus necesidades, siendo esta capaz de construir y entrenar redes neuronales para varios fines como la detección y desciframiento de patrones.

Keras [9] es también una biblioteca de redes neuronales, y es la API ⁴ de alto nivel de la plataforma de TensorFlow, por lo que al instalar TensorFlow incluye la instalación del paquete de Keras, pudiendo usar en primera instancia el paquete de TensorFlow y todas sus dependencias.

⁴Interfaz de Programación de Aplicaciones

Sin embargo, debido a que el paquete importado en el proyecto de Java para el uso del modelo entrenado en Python necesitaba que ese modelo hubiese sido entrenado en la version de Keras 2 o anterior, se ha importado y usado directamente la versión de Keras 2.16.0.

2.1.12. R y RStudio

Para el apartado estadístico del TFG se ha empleado R, se han realizado diversos tests estadísticos usando *R* [21], que es un entorno y lenguaje de programación de código abierto con un enfoque estadístico y es uno de los lenguajes más utilizados en la investigación científica. Se ha usado R para los tests y *RStudio* [30] para editar y ejecutarlos, siendo este un IDE exclusivo para R que destaca principalmente por su enfoque a la computación estadística y gráficos. Rstudio se encarga de instalar las bibliotecas de R que se importan en el código del programa y que no estén instaladas al ejecutar el programa R.

2.1.13. LaTeX y Overleaf

La memoria de este TFG se ha escrito en *LaTeX* [13], que es un lenguaje de texto para la creación de documentos que requieran de una excelente calidad tipográfica. No obstante, entre las opciones posibles para escribirla usando LaTeX, se ha usado uno de los editores de LaTeX más conocidos, *Overleaf* [15], y al ser este un editor online no ha sido necesario instalar en el equipo una distribución de LaTeX.

Los tests programados en R que se mencionan en la sección 2.1.12 generan un documento pdf con los resultados de los test estadísticos ejecutados por el programa. Sin embargo, para poder generar este archivo se requiere de una distribución de LaTeX accesible en el path del sistema. Alternativamente, se puede instalar una librería de LaTeX que contenga lo mínimo posible, y como no hemos requerido de instalar LaTeX para este proyecto ni para la memoria, pues se ha optado por esta opción, eligiendo *TinyTeX* [32] para este cometido.

2.2. Técnicas de IA

En la actualidad existen un montón de técnicas de inteligencia artificial, siendo varias de ellas muy populares y utilizadas en este ámbito. Estas técnicas pueden llegar a proporcionar

a los sistemas desde la capacidad de analizar grandes cantidades de datos, hasta poder ser capaces de identificar patrones ocultos, adaptarse a escenarios nuevos y tomar decisiones cada vez con una precisión mayor. Los autómatas celulares, los algoritmos evolutivos y el machine learning son técnicas bastante conocidas dentro de estas, siendo estas 3 mencionadas sobre las que se va a tratar a continuación.

2.2.1. Autómata celular

La simulación de los transeuntes se podría haber implementado de diferentes maneras, pero una muy conocida es el uso de autómatas celulares. Un autómata celular [5] es un modelo matemático de un sistema dinámico formado por una malla de celdas llamadas células que almacenan distintos estados que son alterados entre un instante y otro en unidades de tiempo discreto, siguiendo intervalos regulares.

Siguiendo esta definición, para simular el movimiento de los agentes el estado almacenado en cada célula indicará si está ocupada por un agente o no, pudiéndose mover un agente a cualquier celda si no está ocupada y si se considera que pertenece a la vecindad de la celda en la que está el agente en ese momento, es decir, que se consideran adyacentes. El problema de esto es que no todas las celdas son espacios transitables para los agentes, ya que en el escenario habrá obstáculos. Debido a esto, cada célula tendrá asignado el tipo de casilla que es, pudiendo ser un obstáculo, una salida o suelo, además del estado almacenado. En caso de que una célula sea un obstáculo, siempre se considerará que está ocupada, y en caso de que sea una salida o suelo, podrá estar ocupada o no dependiendo de si hay algún agente en estas.

Para el autómata celular también se debe considerar el tipo de vecindario, para poder saber cuales se consideran adyacentes a una célula. Hay varios tipos de vecindario, como por ejemplo la vecindad de von Neumann, que considera solo a las casillas que la rodean ortogonalmente como sus vecinas como se puede observar en la figura 1. Para este autómata celular, utilizaremos la vecindad de Moore, que consiste en considerar como vecinas de una celda a las 8 que les rodean tal y como se ve en la figura 2, es decir, a parte de las que le rodean ortogonalmente, también considera a las de las esquinas.

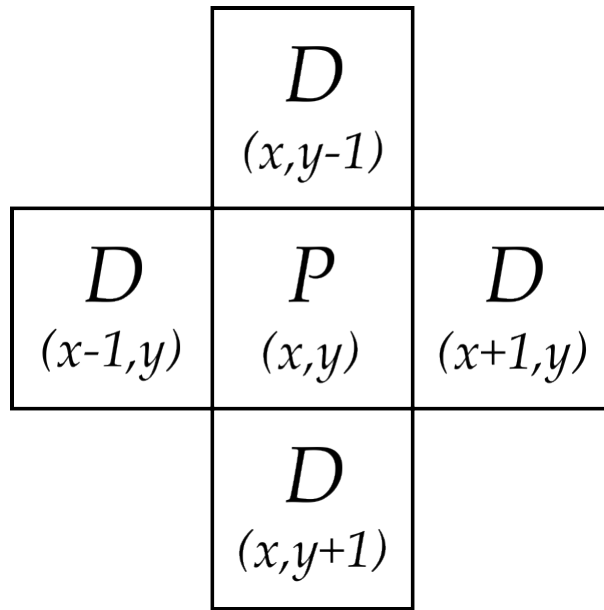


Figura 1: Vecindad de Von Neumann

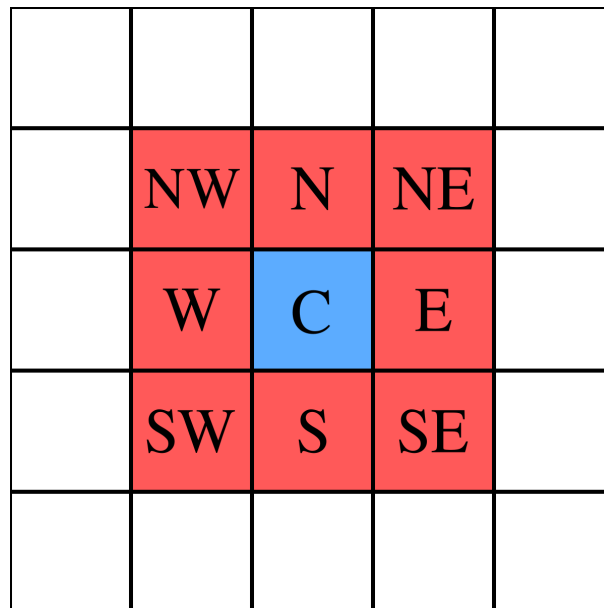


Figura 2: Vecindad de Moore

La probabilidad de que un agente se mueva a una celda vecina disponible se calcula utilizando varios parámetros del agente, pero también depende del espacio celular que utilice el autómata. En este caso, usaremos el algoritmo de Dijkstra con la ayuda de la vecindad de Moore mencionada anteriormente para medir la distancia entre las celdas, asignando la prioridad gracias a esta.

Respecto a los parámetros del agente que se utilizan para calcular la probabilidad del movimiento hacia una celda, son los siguientes los que se tienen en cuenta para esto:

- El sesgo de atracción: este representa con que fuerza los agentes están atraídos a las células de salida.
- La repulsión de la multitud: este indica con que fuerza los agentes son repelidos de las células con mucha multitud.

El comportamiento del autómatas celular que se ha utilizado en este proyecto está descrito con más detalle en el artículo “*Evolutionary Algorithms for Optimizing Emergency Exit Placement in Indoor Environments*” [6].

2.2.2. Algoritmo evolutivo

Una vez se tiene la simulación, se buscará la optimización del escenario distribuyendo las salidas de la forma mejor para que haya la menor cantidad de muertes. Para esto lo que se ha utilizado es un algoritmo evolutivo [6], que es un método de optimización y búsqueda de soluciones a problemas no lineales que se basa en la evolución biológica. Este método empieza con un conjunto de entidades que representan posibles soluciones, las cuales se van mezclando, compitiendo entre sí y evolucionando, y las que sean capaces de mantenerse serán las más aptas.

El proceso usado en este algoritmo es el siguiente:

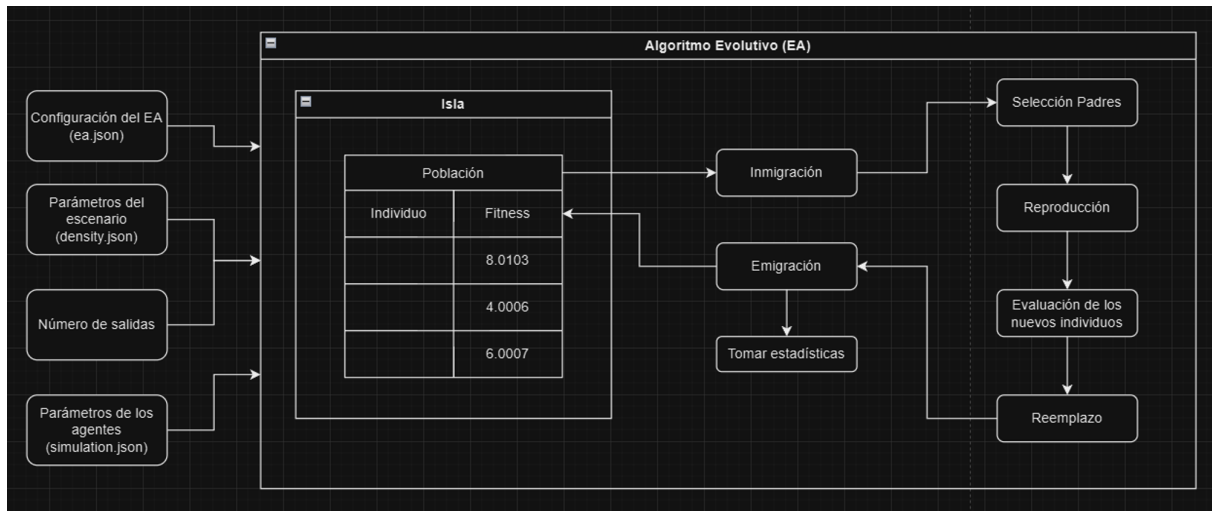


Figura 3: Proceso del algoritmo evolutivo del proyecto

2.2.3. Machine learning

Para entender que es una red neuronal, primero habrá que pasar por el concepto de machine learning [33]. Machine learning es un subcampo de la IA que se centra en el uso de datos y algoritmos para que la IA aprenda ciertas tareas sin la necesidad de instrucciones explícitas. Una red neuronal (artificial) es un programa de machine learning estructurado en capas formadas por neuronas que simula la toma de decisiones del cerebro. Esto nos lleva al concepto de deep learning [4], que las diferencias principales que tiene con el machine learning es que esta tiene más nivel de autonomía, es decir, capacidad de acción, y en lugar de usar redes neuronales simples, utiliza redes neuronales multicapa, también llamadas redes neuronales profundas, las cuales sus neuronas están entrelazadas entre capa y capa.

A partir de la red neuronal que se ha diseñado, una vez se ha realizado el entrenamiento usando datos externos y la red neuronal, se obtiene un modelo funcional. El modelo obtenido se considera machine learning y no deep learning por el aprendizaje supervisado, ya que aunque la red neuronal usada tenga más de una capa y conforme se iba entrenando se volvía mejor, aún necesitaba cierta orientación cada vez que se finalizaba un entrenamiento.

La red neuronal usada se compone de 5 capas, la de entrada, 3 ocultas y la de salida. La capa de entrada está formada por tantas neuronas como número de salidas para un escenario, en este caso, 3. Cada capa oculta se compone de 512 neuronas debido a que es una potencia de

2 y las potencias de 2 anteriores no eran suficientes. Y por último, la capa de salida es de una neurona puesto que queremos que nos devuelva un sólo número.

Para que las capas se activen, es necesario asignarles una función de activación, siendo todas las funciones de activación de esta red neuronal la Unidad Lineal Rectificada (ReLU). Al momento de compilar la red neuronal para generar el modelo, es necesario varios parámetros, como la función de optimización, la de pérdida y la de rendimiento que se suele llamar métrica. Estas son respectivamente el Propagación de Raíz Cuadrática Media (RMSProp), el Error Cuadrático Medio (mse) y el Error Absoluto Medio (mae). Tanto las funciones de activación como los parámetros para compilar el modelo se han asignado siguiendo el prototipo de problema a resolver según el libro “Deep learning with Python” [4].

Una vez compilado, el modelo se entrena usando un conjunto de datos para el entrenamiento y otro conjunto para la validación de este. Estos conjuntos derivan de uno que contiene un montón de conjuntos de puertas, siendo cada uno una solución, junto a su fitness, siendo la calidad de la solución representada mediante un número decimal.

3

Diseño

En este capítulo se definen los requisitos (tanto los funcionales como los no funcionales) y los casos de usos de la herramienta desarrollada en Unity, además de un breve resumen sobre el diseño del EA utilizado un modelo surrogado.

3.1. Actores del sistema

En este proyecto no se ha pensado definir actores ya que no es necesario. Esto es debido a que todas las personas son capaces de usar todas las características, por lo que no es necesario distinguir quien la usa, siendo esta la razón por la que no hay ni registro ni inicio de sesión.

3.2. Requisitos funcionales y no funcionales

Los requisitos de un proyecto de software representan una descripción de las funciones, características y restricciones que debe cumplir el producto final. Dicho con otras palabras, los requisitos definen las condiciones que la aplicación debe tener como mínimo para que se pueda considerar exitoso el proyecto.

3.2.1. Requisitos funcionales

Los requisitos funcionales engloban principalmente las capacidades y comportamientos específicos que se desean tener en un sistema. También pueden llegar a tener en cuenta lo que el sistema no debe hacer. En otras palabras, definen lo que debe hacer un sistema de software.

- **RF 1.** El usuario debe poder cambiar algunos parámetros de la simulación.
 - **RF 1.1** El usuario debe poder cambiar el tamaño de las celdas cuadradas.
 - **RF 1.2** El usuario debe poder cambiar el número de agentes iniciales.

- **RF 1.2** El usuario debe poder cambiar la velocidad de los agentes.
- **RF 2.** El usuario debe poder iniciar una simulación cuando quiera siempre que los campos estén rellenos.
- **RF 3.** El usuario debe poder generar un escenario aleatorio siempre que no se haya iniciado una simulación.
- **RF 4.** El usuario debe poder cambiar la velocidad a la que se reproduce la simulación en todo momento.
- **RF 5.** El usuario debe poder alternar entre una vista desde arriba y otra vista libre.
 - **RF 5.1** El usuario debe poder controlar con total libertad la cámara de la vista libre.
 - **RF 5.2** El usuario debe poder controlar la cámara sin poder girarla.
- **RF 6.** El usuario debe poder cargar a partir de un archivo la traza de los agentes y usarla para una simulación siempre que el escenario para la simulación sea el mismo en el que se ejecutó esa traza.
- **RF 7.** El usuario debe poder cargar a partir de un archivo el escenario en el que se debe ejecutar la simulación.
- **RF 8.** El sistema debe guardar la traza de los agentes en caso de que no se haya cargado una.
- **RF 9.** El sistema debe guardar el escenario en caso de que no se haya cargado uno.

3.2.2. Requisitos no funcionales

A diferencia de los requisitos funcionales, los no funcionales no son las características ni funciones específicas del sistema, sino las propiedades de este, como el rendimiento, la seguridad, la usabilidad y la escalabilidad. Es decir, que describen como debe actuar el sistema de software en lugar de lo que hace el sistema como los funcionales.

- **RNF 1.** El sistema debe estar optimizado para que un ordenador de gama media sea capaz de ejecutarlo sin problemas.
- **RNF 2.** El sistema debe ofrecer al usuario una experiencia cómoda y fluida.

3.3. Casos de uso

Un caso de uso es la descripción de una acción o actividad que un actor puede llevar a cabo para un objetivo específico. La descripción de los casos de usos se compondrán de los siguientes puntos:

- Tabla con algo de información sobre el caso de uso a tratar.
- Escenario principal descrito por pasos.
- Escenarios alternativos en caso de existir.
- Casos de prueba descritos en Gherkin ya que usaremos el enfoque de desarrollo BDD (Behavior Driven Development).

3.3.1. CU 1: Modificación de los parámetros para las simulaciones.

Título	CU 1: Modificación de los parámetros para las simulaciones.
Descripción	El usuario modificará los datos de entrada que serán algunos parámetros de la simulación para cuando la inicie.
Requisito	RF 1
Precondiciones	
Postcondiciones	El usuario ha modificado los parámetros y son visibles los cambios.

Cuadro 1: CU 1: Modificación de los parámetros para las simulaciones.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario modifica alguno de los campos mostrados.
3. El sistema detecta el campo que ha cambiado.
4. El sistema comprueba que el cambio realizado en el campo sea válido.

Escenarios alternativos:

- **Alternativo 1:** El usuario introduce un dato invalido en un campo.

4.a.1 El sistema detecta que el valor introducido en el campo es inválido.

4.a.2 El sistema vacía el campo en el que el valor se ha introducido.

Escenarios de prueba:

- **Feature:** El usuario debe poder modificar los parámetros de entrada de la simulación.
- **Scenario:** Modificación de uno de los parámetros usando un valor válido.

Given El usuario se encuentra en la aplicación.

When El usuario modifica un campo de algún parámetro.

And El valor introducido en el campo es correcto.

Then El sistema mantiene el valor nuevo introducido en el campo.

- **Scenario:** Modificación de uno de los parámetros usando un valor no válido.

Given El usuario se encuentra en el inicio de la aplicación.

When El usuario modifica un campo de algún parámetro.

And El valor introducido en el campo no es válido.

Then El sistema elimina el valor nuevo introducido en el campo.

3.3.2. CU 2: Iniciación de una simulación.

Título	CU 2: Iniciación de una simulación.
Descripción	El usuario podrá iniciar una simulación cuando sea, independientemente de si se está ejecutando otra o no.
Requisito	RF 2
Precondiciones	Los campos de la simulación no están vacíos.
Postcondiciones	El sistema ha iniciado una simulación y el botón de generar escenario se ha desactivado.

Cuadro 2: CU 2: Iniciación de una simulación.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario pulsa el botón de iniciar simulación.
3. El sistema comprueba si hay un escenario cargado.
4. El sistema genera un escenario aleatorio.
5. El sistema comprueba si hay una traza cargada.
6. El sistema genera una traza aleatoria sobre el escenario.
7. El sistema inicia la simulación.

Escenarios alternativos:

- **Alternativo 1:** El usuario inicia la simulación cargando el escenario.

- 3.a.1 El sistema detecta que hay un escenario cargado y lo crea.
- 3.a.2 El sistema comprueba si hay una traza cargada.
- 3.a.3 El sistema genera una traza aleatoria sobre el escenario.
- 3.a.4 El sistema inicia la simulación.

- **Alternativo 2:** El usuario inicia la simulación cargando el escenario y la traza.

- 3.b.1 El sistema detecta que hay un escenario cargado y lo crea.
- 3.b.2 El sistema comprueba si hay una traza cargada.
- 3.b.3 El sistema detecta que hay una traza cargada y comprueba si el escenario es el mismo que el de la traza.
- 3.b.4 El sistema crea los agentes.
- 3.b.5 El sistema inicia la simulación.

- **Alternativo 3:** El sistema falla al iniciar la simulación cargando el escenario y la traza.

- 3.b.3.c.1 El sistema detecta que no coinciden el escenario cargado con el de la traza y para el proceso.

- **Alternativo 4:** El usuario inicia la simulación cargando la traza.

5.d.1 El sistema detecta que hay una traza cargada y crea los agentes.

5.d.2 El sistema comprueba si hay una traza cargada.

5.d.3 El sistema detecta que hay una traza cargada y comprueba si el escenario es el mismo que el de la traza.

5.d.4 El sistema detecta que no coinciden el escenario con el de la traza y para el proceso.

Escenarios de prueba:

- **Feature:** El usuario inicia una simulación.

- **Scenario:** El usuario inicia una simulación sin cargar ningún archivo.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón para iniciar la simulación

And Está marcada la opción para guardar la traza.

And Está marcada la opción para guardar el escenario.

Then El sistema inicia una simulación aleatoria.

- **Scenario:** El usuario inicia la simulación cargando el escenario.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón para iniciar la simulación

And Está marcada la opción para guardar la traza.

And Hay un escenario cargado.

Then El sistema inicia una simulación con agentes aleatorios.

- **Scenario:** El usuario inicia la simulación cargando el escenario y la traza.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón para iniciar la simulación

And Hay una traza cargada.

And Hay un escenario cargado.

And El escenario cargado coincide con el de la traza.

Then El sistema inicia una simulación con agentes aleatorios.

- **Scenario:** El sistema falla al iniciar la simulación cargando el escenario y la traza.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón para iniciar la simulación

And Hay una traza cargada.

And Hay un escenario cargado.

And El escenario cargado no coincide con el de la traza.

Then El sistema para el proceso de iniciar una simulación.

- **Scenario:** El usuario inicia la simulación cargando la traza.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón para iniciar la simulación

And Hay una traza cargada.

And Está marcada la opción para guardar el escenario.

Then El sistema para el proceso de iniciar una simulación.

3.3.3. CU 3: Generación de un escenario aleatorio.

Título	CU 3: Generación de un escenario aleatorio.
Descripción	El usuario podrá generar un escenario aleatorio cuando no haya una simulación en ejecución.
Requisito	RF 3
Precondiciones	El sistema no puede estar ejecutando ninguna simulación. El campo del tamaño de celdas no puede estar vacío.
Postcondiciones	El sistema ha generado el escenario aleatorio.

Cuadro 3: CU 3: Generación de un escenario aleatorio.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario pulsa el botón de generar escenario.
3. El sistema genera un escenario aleatorio.

Escenarios de prueba:

- **Feature:** Generación de un escenario aleatorio.
- **Scenario:** El usuario genera un escenario aleatorio.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón para generar el escenario.

And No hay una simulación en ejecución.

And Todos los campos están rellenos.

Then El sistema inicia una simulación aleatoria.

3.3.4. CU 4: Modificación de la velocidad de la simulación.

Título	CU 4: Modificación de la velocidad de la simulación.
Descripción	El usuario podrá modificar la velocidad de reproducción de la simulación en ejecución en cualquier momento.
Requisito	RF 4
Precondiciones	
Postcondiciones	

Cuadro 4: CU 4: Modificación de la velocidad de la simulación.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario cambia el valor de la velocidad de reproducción mediante un slider.

3. El sistema cambia la velocidad de todos los agentes.

Escenarios de prueba:

- **Feature:** Modificación de la velocidad de la simulación.
- **Scenario:** El usuario modifica la velocidad de reproducción de la simulación.

Given El usuario se encuentra en la aplicación.

When El usuario arrastra el slider de la velocidad.

Then El sistema cambia la velocidad de todos los agentes.

3.3.5. CU 5: Alternar entre las dos vistas posibles.

Título	CU 5: Alternar entre las dos vistas posibles.
Descripción	El usuario podrá alternar entre dos vistas distintas para observar la simulación. Cada vista tendrá una cámara distinta con sus respectivos controles.
Requisito	RF 5
Precondiciones	
Postcondiciones	

Cuadro 5: CU 5: Alternar entre las dos vistas posibles.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario pulsa el check de cambiar de vista.
3. El sistema cambia la vista.

Escenarios de prueba:

- **Feature:** Alternar entre las dos vistas.
- **Scenario:** Cambiar de vista desde arriba a vista libre.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el check de cambiar de vista

And La vista actual es la de arriba.

Then El sistema cambia la vista a vista libre.

- **Scenario:** Cambiar de vista libre a vista desde arriba.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el check de cambiar de vista

And La vista actual es la libre.

Then El sistema cambia la vista a vista desde arriba.

3.3.6. CU 6: Cargar una traza externa.

Título	CU 6: Cargar una traza externa.
Descripción	El usuario podrá escoger un archivo json con la traza y ejecutarla en una simulación siempre que se haya cargado el escenario y coincida con este.
Requisito	RF 6
Precondiciones	
Postcondiciones	El sistema muestra la ruta en la que está almacenada el archivo.

Cuadro 6: CU 6: Cargar una traza externa.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario desactiva el check de guardar traza.
3. El usuario pulsa el botón de cargar traza.
4. El sistema abre un explorador de archivos.
5. El usuario escoge el archivo a cargar.

6. El sistema guarda la ruta del archivo escogido.

Escenarios alternativos:

- **Alternativo 1:** El usuario cancela la búsqueda del archivo de la traza.

5.a.1 El usuario pulsa el botón de cancelar o de cerrar ventana del explorador de archivos.

5.a.2 El sistema cierra el explorador de archivos.

Escenarios de prueba:

- **Feature:** Cargar una traza externa.
- **Scenario:** El usuario sube una traza a la herramienta.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de cargar traza.

And El usuario selecciona un archivo json.

Then El sistema guarda la ruta del archivo.

- **Scenario:** El usuario cancela la búsqueda del archivo de la traza.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de cargar traza.

And El usuario cierra el explorador de archivos mediante el botón de cancelar o el de cerrar ventana.

Then El sistema no cambia la ruta del archivo.

3.3.7. CU 7: Cargar un escenario externo.

Título	CU 7: Cargar un escenario externo.
Descripción	El usuario podrá escoger un archivo json con el escenario y usarlo para una simulación.
Requisito	RF 7
Precondiciones	
Postcondiciones	El sistema muestra la ruta en la que está almacenada el archivo.

Cuadro 7: CU 7: Cargar un escenario externo.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario desactiva el check de guardar escenario.
3. El usuario pulsa el botón de cargar escenario.
4. El sistema abre un explorador de archivos.
5. El usuario escoge el archivo a cargar.
6. El sistema guarda la ruta del archivo escogido.

Escenarios alternativos:

- **Alternativo 1:** El usuario cancela la búsqueda del archivo del escenario.

5.a.1 El usuario pulsa el botón de cancelar o de cerrar ventana del explorador de archivos.

5.a.2 El sistema cierra el explorador de archivos.

Escenarios de prueba:

- **Feature:** Cargar un escenario externo.
- **Scenario:** El usuario sube un escenario a la herramienta.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de cargar escenario.

And El usuario selecciona un archivo json.

Then El sistema guarda la ruta del archivo.

- **Scenario:** El usuario cancela la búsqueda del archivo del escenario.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de cargar escenario.

And El usuario cierra el explorador de archivos mediante el botón de cancelar o el de cerrar ventana.

Then El sistema no cambia la ruta del archivo.

3.3.8. CU 8: Guardar la traza que se ejecutará en la simulación.

Título	CU 8: Guardar la traza que se ejecutará en la simulación.
Descripción	El usuario podrá guardar la traza ejecutada en la simulación en un archivo json.
Requisito	RF 8
Precondiciones	
Postcondiciones	El sistema muestra la ruta en la que se almacenará el archivo.

Cuadro 8: CU 8: Guardar la traza que se ejecutará en la simulación.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario activa el check de guardar traza.
3. El usuario pulsa el botón de guardar traza.
4. El sistema abre un explorador de archivos.

5. El usuario escoge la ubicación donde se guardará el archivo.
6. El sistema guarda la ruta escogida para el archivo.

Escenarios alternativos:

- **Alternativo 1:** El usuario cancela la búsqueda de la ubicación para la traza.

5.a.1 El usuario pulsa el botón de cancelar o de cerrar ventana del explorador de archivos.

5.a.2 El sistema cierra el explorador de archivos.

Escenarios de prueba:

- **Feature:** Guardar la traza que se ejecute en la simulación.
- **Scenario:** El usuario elige guardar la traza ejecutada en la simulación.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de guardar traza.

And El usuario selecciona una ubicación.

Then El sistema guarda la ruta para el archivo.

- **Scenario:** El usuario cancela la búsqueda de la ubicación para guardar la traza.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de guardar escenario.

And El usuario cierra el explorador de archivos mediante el botón de cancelar o el de cerrar ventana.

Then El sistema no cambia la ruta para el archivo.

3.3.9. CU 9: Guardar el escenario que se usará en la simulación.

Título	CU 9: Guardar el escenario que se usará en la simulación.
Descripción	El usuario podrá guardar el escenario utilizado en la simulación en un archivo json.
Requisito	RF 9
Precondiciones	
Postcondiciones	El sistema muestra la ruta en la que se almacenará el archivo.

Cuadro 9: CU 9: Guardar el escenario que se usará en la simulación.

Escenario principal:

1. El usuario se encuentra en la aplicación.
2. El usuario activa el check de guardar escenario.
3. El usuario pulsa el botón de guardar escenario.
4. El sistema abre un explorador de archivos.
5. El usuario escoge la ubicación donde se guardará el archivo.
6. El sistema guarda la ruta escogida para el archivo.

Escenarios alternativos:

- **Alternativo 1:** El usuario cancela la búsqueda de la ubicación para el escenario.
 - 5.a.1 El usuario pulsa el botón de cancelar o de cerrar ventana del explorador de archivos.
 - 5.a.2 El sistema cierra el explorador de archivos.

Escenarios de prueba:

- **Feature:** Guardar el escenario en el que se ejecute la simulación.
- **Scenario:** El usuario elige guardar el escenario utilizado en la simulación.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de guardar escenario.

And El usuario selecciona una ubicación.

Then El sistema guarda la ruta para el archivo.

- **Scenario:** El usuario cancela la búsqueda de la ubicación para guardar el escenario.

Given El usuario se encuentra en la aplicación.

When El usuario pulsa el botón de guardar escenario.

And El usuario cierra el explorador de archivos mediante el botón de cancelar o el de cerrar ventana.

Then El sistema no cambia la ruta para el archivo.

3.4. Diseño del modelo surrogado

Como se ha mencionado anteriormente, la segunda parte de este proyecto consiste en optimizar un algoritmo evolutivo sustituyendo el autómatas celular utilizado para calcular el fitness por un modelo surrogado entrenado usando machine learning. A continuación, se comentará brevemente el diseño del EA utilizando un modelo surrogado puesto que en la sección 2.2 se ha comentado todo esto.

El EA desarrollado busca la distribución de las salidas más óptima para un escenario tal que se mueran el menor número de agentes posible. Dependiendo de lo que se busque, cada EA sigue un proceso de selección de padres, reproducción y reemplazo, siendo en nuestro caso el proceso mostrado gráficamente en la figura 3.

El objetivo del modelo surrogado entrenado es sustituir el cálculo del fitness para que su coste computacional sea mucho menor, por lo que en el proceso del EA descrito en la figura 3, bastaría con sustituir en el paso de la evaluación de los nuevos individuos la ejecución del autómatas celular por la llamada a la función del modelo surrogado.

La estructura y diseño del modelo surrogado necesarias para entrenar este modelo están descritas ya se han descrito con anterioridad, sin embargo, no se ha tratado cómo se va a programar. La programación del entrenamiento es facilitada utilizando Tensorflow y Keras, las

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=512, activation='relu', input_shape=(train_data.shape[1],)))
model.add(keras.layers.Dense(units=512, activation='relu'))
model.add(keras.layers.Dense(units=512, activation='relu'))
model.add(keras.layers.Dense(1))
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Figura 4: Capas del modelo implementads utilizando Keras

cuales son tecnologías descritas en la sección 2.1.11, pudiendo observar la estructura descrita en la sección 2.2.3 en la siguiente figura.

4

Implementación

Este apartado consiste en la explicación de las arquitecturas y modelos que sigue el sistema realizado para la herramienta desarrollada de este proyecto, y las razones por las que se han usado estas en lugar de otras opciones.

4.1. Arquitectura de la herramienta

En la actualidad existen bastantes tipos de arquitecturas, desde las que se pueden usar en varias situaciones para varios tipos de aplicaciones hasta las que están pensadas para casos más específicos. Un ejemplo es la arquitectura cliente-servidor, que consiste en repartir el trabajo entre los clientes y los servidores. Está pensada para un sistema operativo multiusuario distribuido a través de una red de computadoras, aunque también se puede aplicar para programas que se ejecuten en un solo ordenador.

Debido a que la herramienta de simulación debe estar atento a que acción realiza el usuario en todo momento, independientemente de si se está ejecutando la simulación o no, la arquitectura principal que se ha usado para esta aplicación es una basada en notificaciones. El beneficio más destacable que tiene esta arquitectura es el asincronismo que proporciona para poder realizar al instante las tareas (*callbacks*) que encarga el usuario. Para cumplir esto se hace uso de los delegados, eventos y acciones de Unity aplicando el patrón de diseño llamado el patrón observador (*Observer*) en más de una ocasión, como al momento de saber cuando se abre el explorador de archivos y cuando se selecciona la ruta en estos, para saber cuando se cambia de vista, y para cuando se usa cualquier elemento de la interfaz que necesita notificar al momento, como los botones o la barra que indica la velocidad de la simulación.

Un delegado es una referencia a un método, permitiéndonos tratar ese método como un tipo y pasarlo como variable para un callback, devolviendo ese tipo.

Un evento es prácticamente igual que un delegado multicast, es decir, un delegado que

puede tener más de un método asignado a él. Al igual que un delegado, cuando se invoque, notificará a todos los métodos que le hacen referencia con lo especificado en la declaración del evento.

Una acción está basado en un delegado, pero a diferencia de estos y de los eventos, no devuelve ningún tipo. Estos tienen la ventaja de que en C# son los más sencillos de declarar.

Además del patrón observador, se han utilizado otros patrones de diseño, como el patrón constructor (Builder) para lo siguiente:

- Los parámetros del autómatas celular se reciben mediante una clase inicializada mediante una clase Builder, ya que podemos asignarle los parámetros que queramos desde el principio, y aunque esto al final no sea necesario, ya que valdría con dejar por defecto los valores que no se reciben gracias al usuario, era una opción que se contemplaba y se sigue contemplando de cara al futuro.
- Al igual que este patrón se ha utilizado para los parámetros del autómatas celular, también se han usado para los parámetros de los agentes por la misma razón.

4.2. Diagrama de clases

Para poder entender mejor las clases que componen la herramienta desarrollada y las relaciones que existen entre estas clases, se ha separado el diagrama de clases del código de la herramienta en varios diagramas de clases según los paquetes en los que se encuentren las clases para que esté más estructurado, representados en varias figuras a continuación.

En la figura 5 se describe como se construye y funciona el autómatas celular que es representado con la clase “CellularAutomaton”. Esta es la que se encarga del comportamiento del autómatas celular y de hacer que todos los agentes se muevan cuando es debido, y para esto, se ha creado una clase estática interna “ListExtensions” para la gestión de la lista de los agentes que no han salido del escenario. Se puede observar que los parámetros del autómatas celular se inicializan gracias a un patrón constructor, que a su vez se compone de otros constructores que son necesarios. Esto se ha implementado así ya que había la posibilidad de añadir en un futuro otros parámetros extras que no fuesen opcionales. También nos muestra que se ha utilizado un record llamado “Location” para la localización de las celdas y para el almacenamiento del camino que han seguido los agentes debido a que estos no varían de localización.

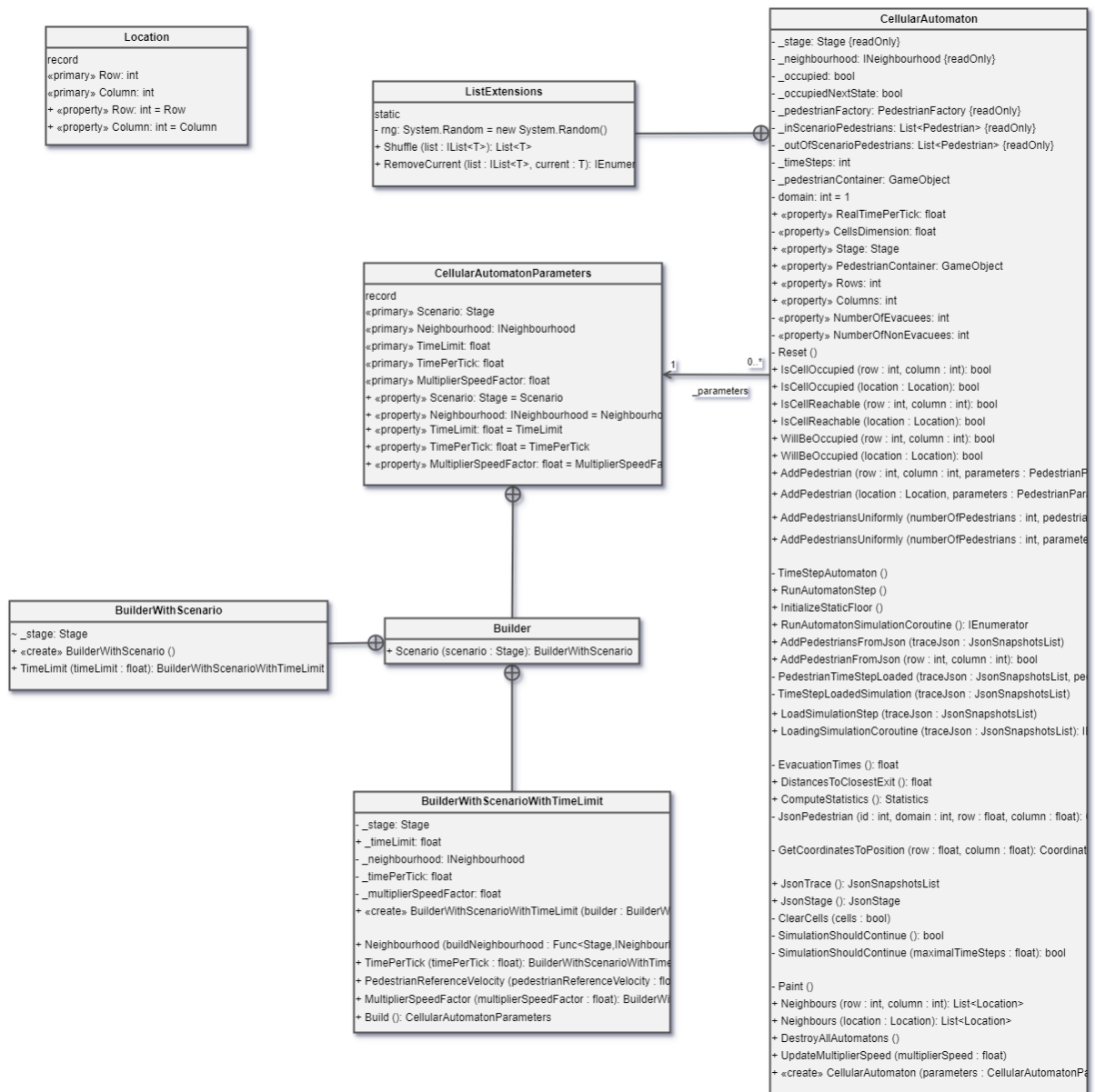


Figura 5: Estructura del Autómata celular

La figura 6 muestra las dos maneras de crear un escenario, desde un JSON o generar uno aleatorio. Como los pasos de los escenarios generados de las 2 formas son los mismos pero difieren en el proceso de algunos pasos, se ha pensado en realizar una clase por cada forma de generar el escenario que herede de la clase abstracta “Stage”. Un escenario se compone de varias células “Cell”, y tal como se explica en la sección 2.2.1, cada célula puede ser un obstáculo, una salida o suelo, siendo esto representado por el enumerado “CellTypeEnum”.

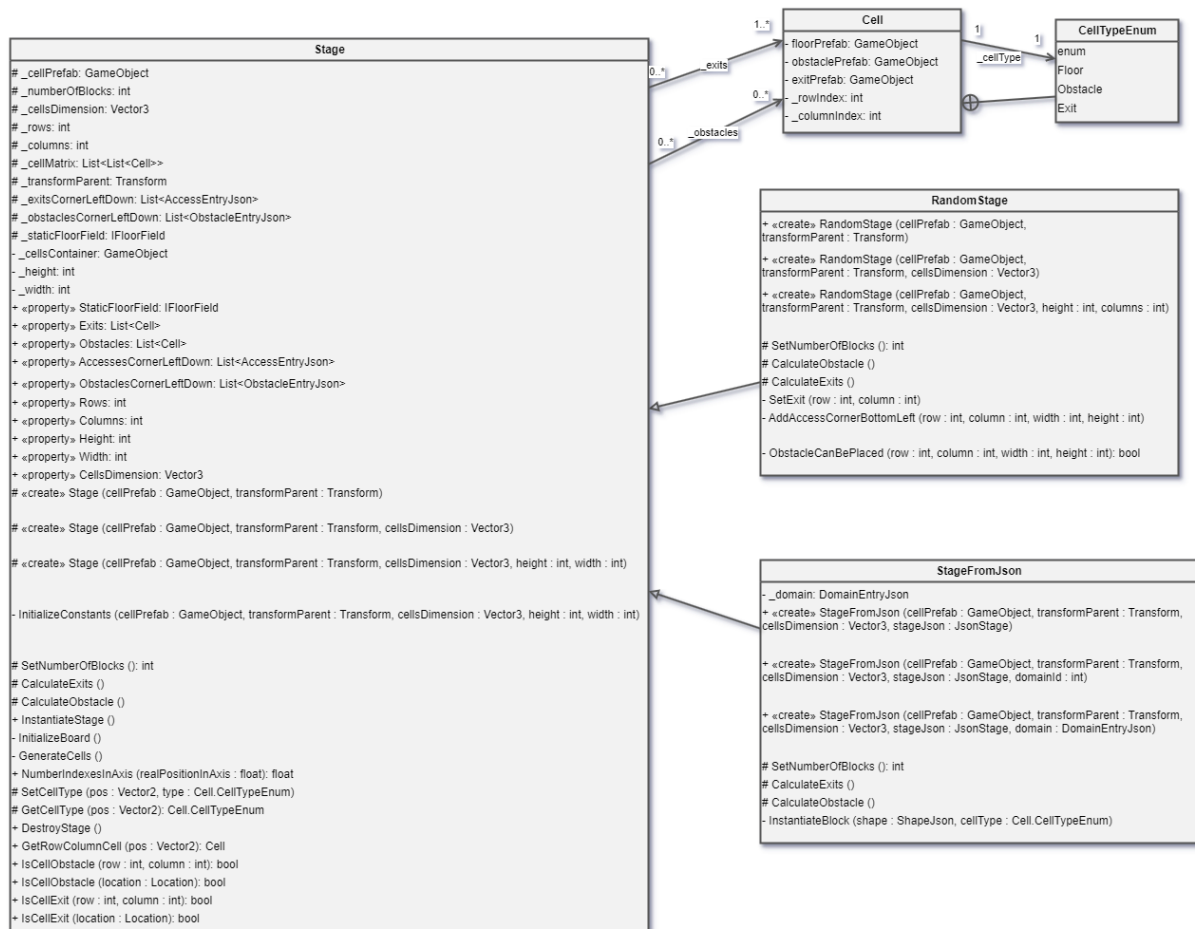


Figura 6: Estructura del escenario

Además de lo anterior, en el apartado 2.2.1 también se menciona que es necesario calcular la distancia entre las celdas. Esto es necesario para que los movimientos de los agentes dependan de esta distancia, haciendo que las celdas almacenen un orden de prioridad mediante el record “MyNode” y que se calcula usando la clase abstracta “StaticFloorField”, ambas clases mostradas en la figura 7. Usaremos el algoritmo de Dijkstra representado en la clase “DijkstraS-taticFloorField” que hereda de la clase anterior. A su vez, se ha creado una clase “DijkstraS-taticFloorFieldWithMooreNeighbourhood” que vuelve a heredar de la anterior mencionada puesto que nos apoyaremos en la vecindad de Moore usando la clase “MooreNeighbourhood” que aparece en la figura 8. Aunque se haya creado la clase “VonNeumannNeighbourhood” debido a que existía la posibilidad de usarla, no se llega a usar en ningún momento.

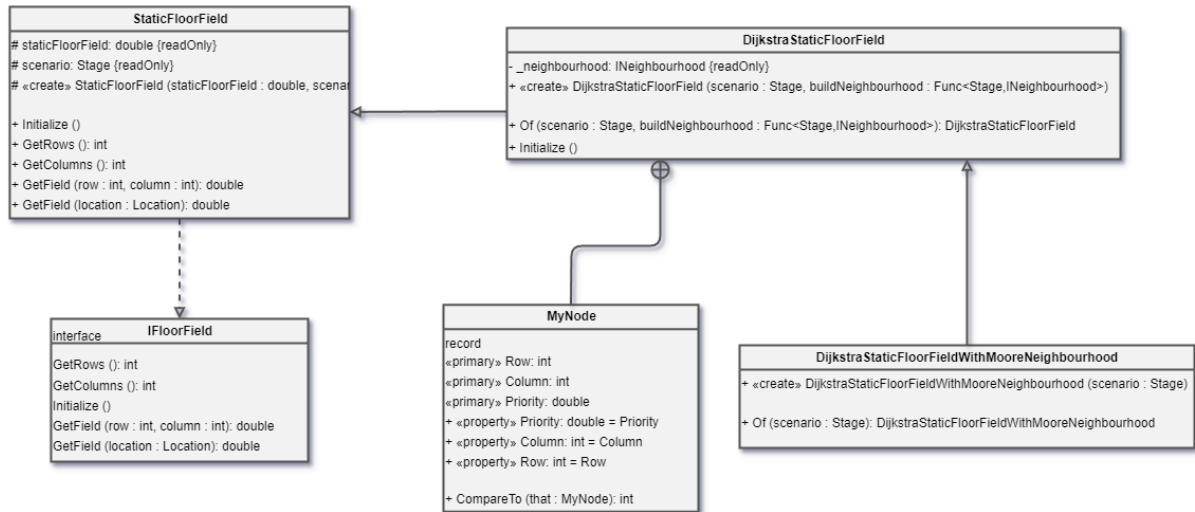


Figura 7: Estructura del Floor Field

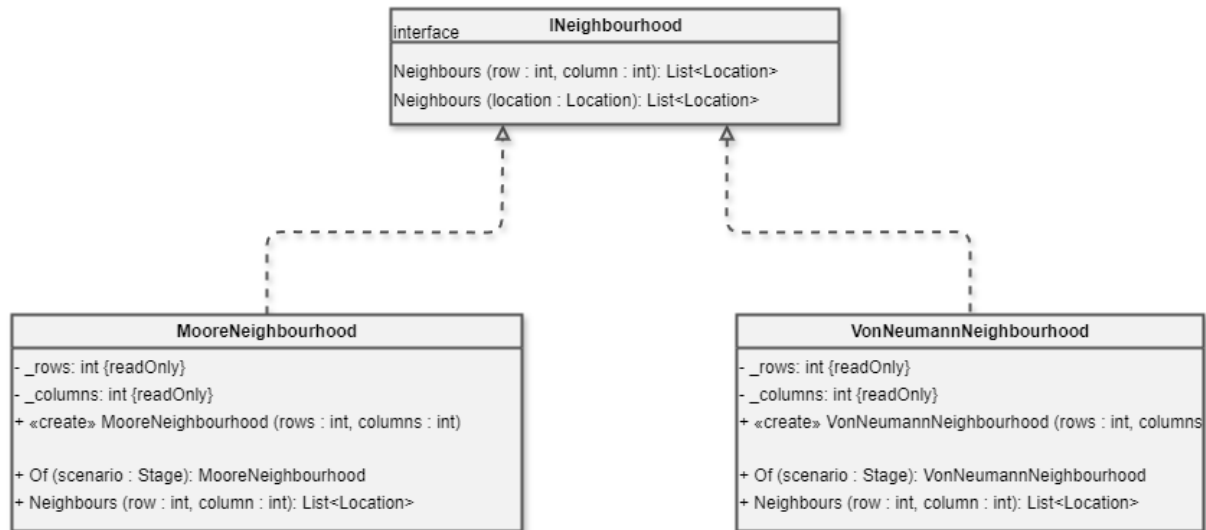


Figura 8: Estructura de las casillas vecinas

El automata celular contiene todos los agentes, estando la estructura de los agentes representado en la figura 9 en la que se puede ver que tienen varios parámetros de los que dependerá su próximo movimiento. Algunos de estos parámetros se pensaron para que se pudiesen modificar desde la simulación de manera voluntaria, por lo que el record “PedestrianParameters” que contiene esos parámetros se construyen utilizando un patrón constructor. El siguiente

movimiento que vaya a realizar un agente se elige de entre una lista de movimientos provisionales, representado cada uno con el record “TentativeMovement” definido dentro de la clase de los agentes llamada “Pedestrian”. Por último, para crear los agentes se ha utilizado un patrón factoría, y aunque la factoría representada por la clase “PedestrianFactory” no herede de ninguna interfaz, se puede considerar como el patrón factoría ya que no se ha creado la interfaz a propósito puesto que se sabía que no habría otros tipos de agentes.

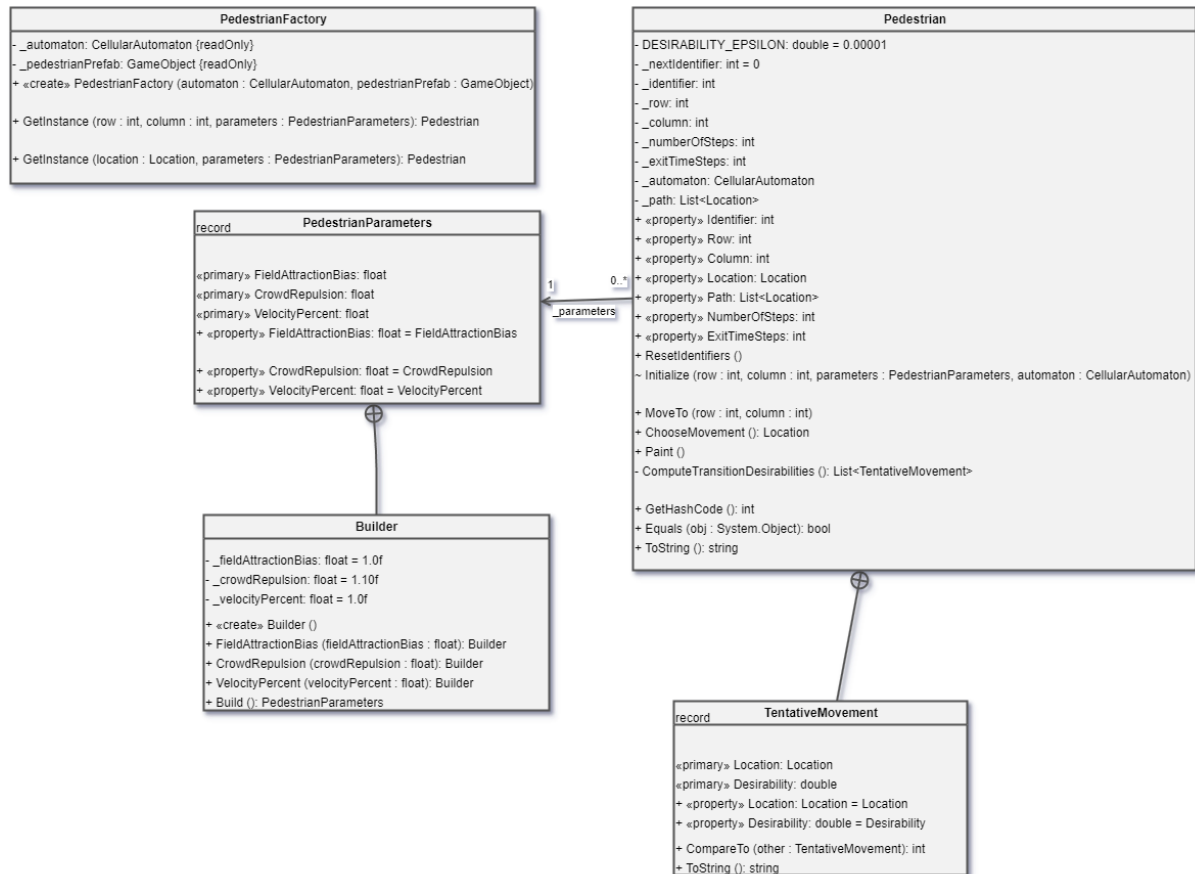


Figura 9: Estructura de los agentes

Para poder cargar y guardar los archivos JSON tanto el escenario como de la traza que almacena la posición inicial y los movimientos de cada agente, se ha utilizado la clase estática “SaveJsonManager” mostrada en la figura 10 con métodos que hacen posible estas funcionalidades. Aunque estos métodos no dependan de una variable para poder usarlos, si requieren de argumentos que almacenen los datos necesarios. Para cargar ambos JSON no hay problema, ya que solo es necesario la ruta del archivo. Sin embargo, para guardarlos, se requiere de las

clases específicas que representan los datos en los JSON.

En caso de querer guardar una traza, se requiere de la lista de instantes de tiempo, siendo cada instante un objeto del tipo “DomainEntryJson” que contiene a la multitud y el tick en ese momento, y también de la dimensión de las células representadas mediante un número decimal. El método para guardar el JSON puede recibir ambos argumentos por separado o puede recibir la clase “JsonSnapshotsList” que contiene ambos argumentos. Esta estructura está representada completamente en la figura 11 junto al resto de sus dependencias.

En caso de querer guardar un escenario, se requiere de la lista de dominios, siendo cada dominio un objeto del tipo “DomainEntryJson” que contiene los obstáculos y salidas, y también de la lista de pasarelas, que son un conjunto de identificadores guardados en la clase “GatewayEntryJson”. Ambas clases se pueden utilizar por separado o también se puede usar la clase “JsonStage”, que contiene ambas listas. Esta estructura está representada completamente en la figura 12 junto al resto de sus dependencias, que entre ellas se encuentra la clase “ShapeType” mostrada en la figura 13 que indica la forma que tiene el conjunto de entradas o salidas al que pertenece esa célula.

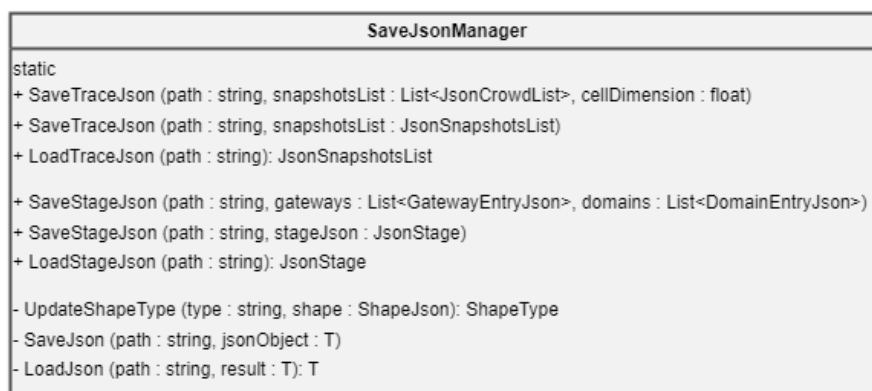


Figura 10: Clase para cargar y guardar los JSON

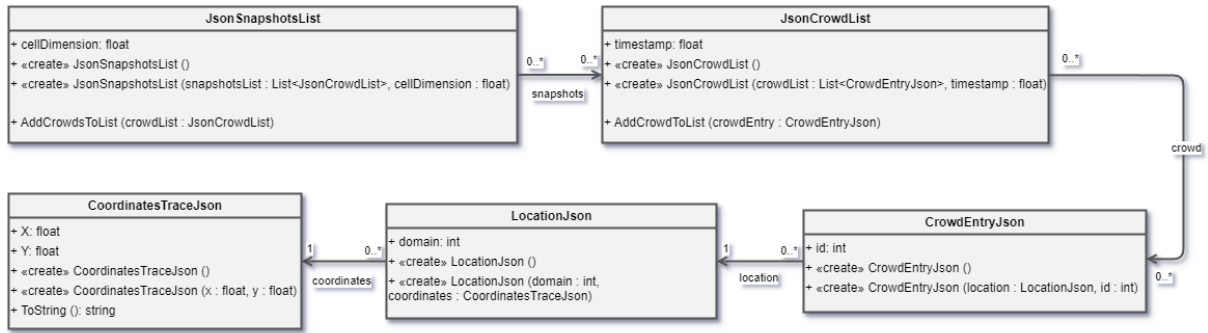


Figura 11: Estructura para los archivos JSON de las trazas

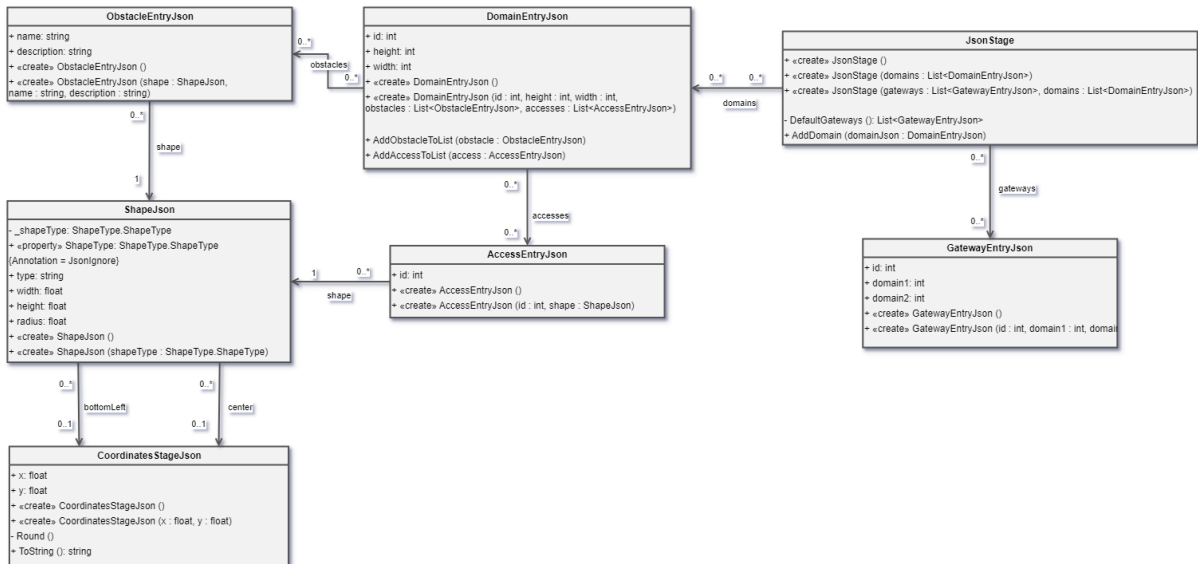


Figura 12: Estructura para los archivos JSON de los escenarios

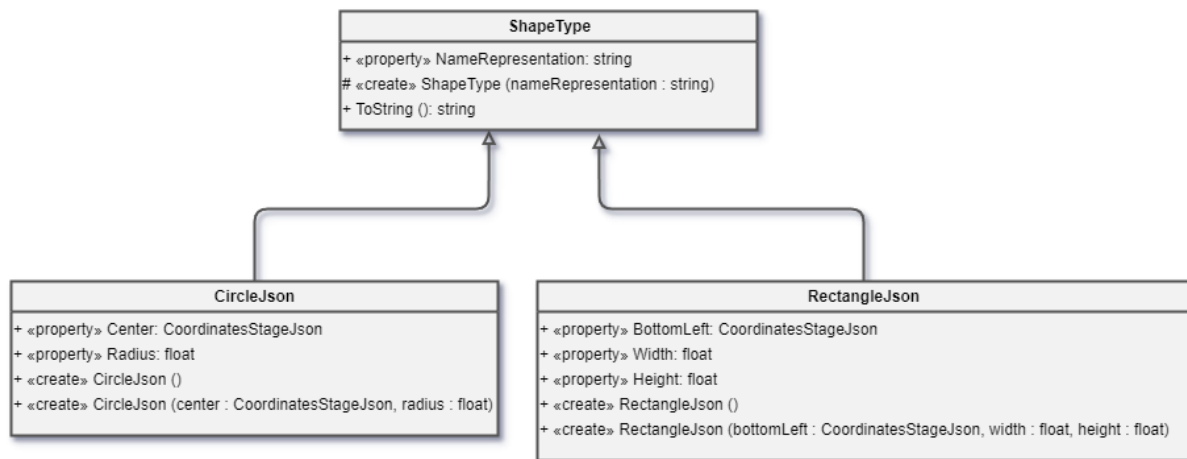


Figura 13: Tipos de formas en el Json del escenario para los obsáculos y puertas

La UI⁵ se maneja gracias a un objeto situado en la escena de Unity que hace de controlador de la interfaz y que solo contiene la clase “UIManager” de la figura 14. Esta clase debe estar en la escena puesto que se necesita referenciar dentro de la clase a los elementos de la UI para poder añadirle los escuchadores o mejor conocidos como listeners. A parte de este controlador y el del JSON que se explica a continuación, hay otros controladores en la escena mostrados en la figura 15 como “InitializateStage”, que es el del escenario y la simulación, y “CameraController”, siendo este el de la cámara.

Como se ha mencionado antes, además del controlador de la UI, también se muestra la estructura del controlador de los JSON en la figura 14. Para actualizar el texto de las rutas de los json, se ha optado por crear la clase “JsonPathTextUI” que se encarga de actualizar el texto en caso de que se haya modificado la ruta del archivo que le corresponde. Esto se maneja gracias al argumento del tipo enumerado “TypeJsonButton” que indica si es la ruta de la traza o del escenario la que se ha modificado, y a un argumento booleano que es verdadero en caso de que sea ruta para guardar el archivo y falso en caso de que sea la ruta para cargarlo. La clase que falta por mencionar de la figura 14, con nombre “PathsForJson”, es la que se encarga de gestionar el explorador de archivos y del almacenamiento de las rutas de guardado y cargado para los archivos.

⁵Interfaz de Usuario

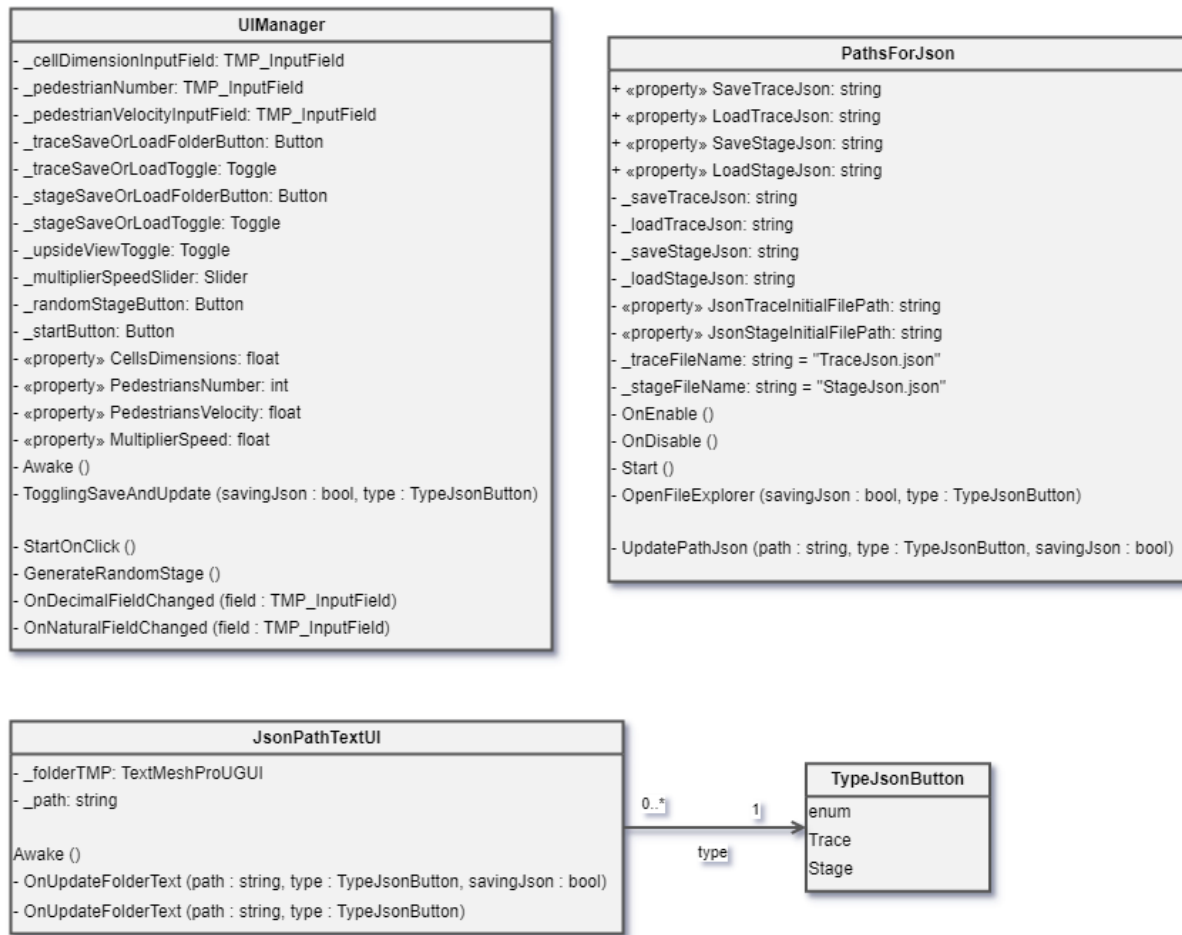


Figura 14: Controlador para la UI y las rutas para los JSON

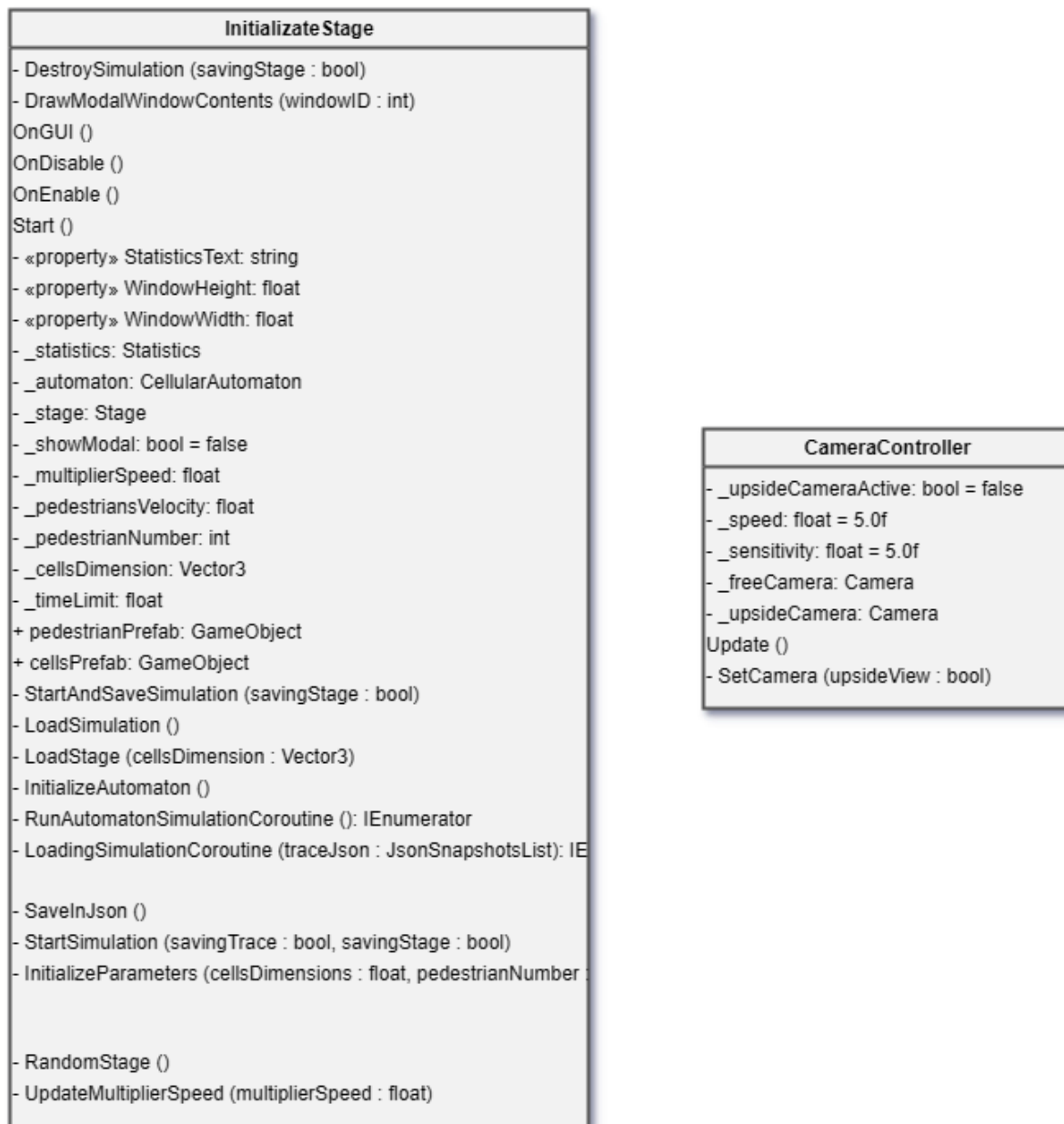


Figura 15: Controlodares para la escena en Unity

A pesar de que hay un controlador de la interfaz para manejar los elementos de esta, lo idóneo es que la funcionalidad se maneje mediante eventos y listeners ya que de esta forma no es necesario referenciar a las clases que implementen esa funcionalidad. Se han usado los 3 tipos de eventos mostrados en la figura 16, los de la cámara que están dentro de la clase estática “CameraEvents”, los del explorador de archivos en la clase estática “FileExplorerEvents” y los

eventos de la generación del escenario y la simulación en la clase estática “SimulationEvents”.

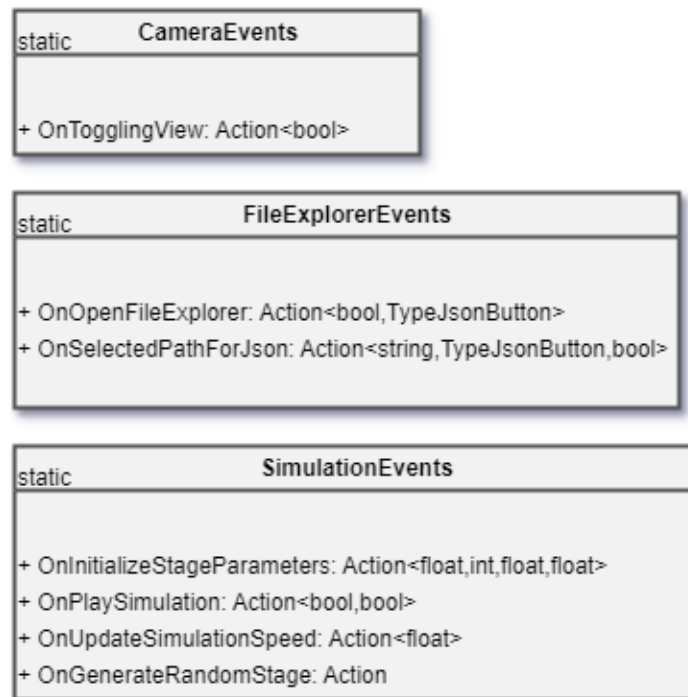


Figura 16: Tipos de eventos de la herramienta

Una vez que termina la simulación, se calculan las estadísticas de la simulación usando el record “Statistics” que aparece en la figura 17 y se muestran en la pantalla.

record	Statistics
<pre> «primary» MeanSteps: float «primary» MeanEvacuationTime: float «primary» MedianSteps: float «primary» MedianEvacuationTime: float «primary» NumberOfEvacuees: int «primary» NumberOfNonEvacuees: int + «property» MeanSteps: float = MeanSteps + «property» MeanEvacuationTime: float = MeanEvacuationTi + «property» MedianSteps: float = MedianSteps + «property» MedianEvacuationTime: float = MedianEvacuati + «property» NumberOfEvacuees: int = NumberOfEvacuees + «property» NumberOfNonEvacuees: int = NumberOfNonEv + Bernoulli (successProbability : float): bool + Discrete (collection : List<T>, desirability : Func<T,double>): + Mean (data : int): float + Mean (data : float): float + Median (data : int): float + Median (data : float): float </pre>	

Figura 17: Clase para las estadísticas de una simulación

5

Pruebas

Las pruebas son un proceso importante a la hora de desarrollar una herramienta. Estas sirven principalmente para comprobar que la aplicación esté cumpliendo sus objetivos y que todos los elementos de esta funcionen correctamente.

Debido a que a lo largo del proceso de desarrollo he ido aprendiendo sobre Unity, al principio desconocía sobre si existía algún framework o herramienta que facilitase la ejecución de tests automatizados. A mitad del desarrollo, conocí la herramienta Unity Test Framework (UTF) [28], el cual es un framework que nos permite realizar tests automatizados y ejecutarlos tanto en el modo de edición como cuando se está reproduciendo la escena de Unity. UTF se integra con NUnit, que es una de las biblioteca de tests de código abierto para .NET más conocidas que hay. Sin embargo, como el coste de aprender dos tecnologías mientras se seguía desarrollando era elevado, además de haber ido probando manualmente hasta ese momento si las funciones que se habían terminado de desarrollar, se llegó a la conclusión de que no era estrictamente necesario el uso de este framework.

En el caso de la segunda parte del proyecto, el proceso para modificar los archivos de datos obtenidos y entrenar el modelo de machine learning ha sido también comprobándolo manualmente. El proceso para entrenar el modelo ha sido principalmente de prueba y error, empezando este proceso con unos parámetros que son genéricos y que sirven para varios casos, incluyendo el objetivo de este proyecto, y acabando con parámetros más específicos, obteniendo mejores resultados que en un principio. Este proceso se ha facilitado obteniendo algunos gráficos después de cada modelo para poder intuir que parámetros cambiar para que el siguiente modelo obtenga mejores resultados que el último entrenado.

Por tanto, no se llegó a realizar ningún tipo de pruebas automatizadas para el desarrollo de este proyecto.

5.1. Pruebas de usabilidad

Al ser la herramienta desarrollada con Unity una realizada para un proyecto de investigación, los usuarios que la van a usar son los mismos que la han ideado, por lo que las pruebas de usabilidad han consistido en que en cada iteración, el tutor verificase si el desarrollo hasta el momento iba según lo previsto, que todo funcionase correctamente y que sugiriese cambios y funcionalidades nuevas que no estuviesen pensadas con anterioridad.

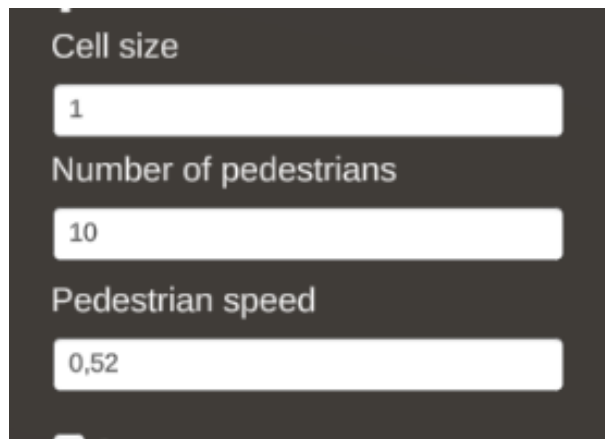
6

Resultados

Esta sección muestra los resultados obtenidos de la herramienta desarrollada en Unity además de servir como manual para mostrar el cómo utilizarla. Aunque su uso se lleve a cabo en una sola pantalla y no sea compleja, en este apartado se muestra descripciones e instrucciones para comprender de una forma más sencilla el como utilizarla.

6.1. Modificar parámetros

La herramienta solo tiene una única pantalla, la cual se ve al iniciarla. Lo primero que el usuario puede observar en esta pantalla es una interfaz a la derecha con elementos interactivos. Entre estos elementos se encuentran tal y como se puede ver en la figura 18 los campos rellenables con números que les dan los valores a ciertos parámetros de la simulación. Para modificarlos basta con hacer click en el campo correspondiente que se quiera modificar y escribir el número.



The image shows a dark-themed user interface for a simulation. It contains three labeled input fields, each with a white text box. The first field is labeled 'Cell size' and contains the number '1'. The second field is labeled 'Number of pedestrians' and contains the number '10'. The third field is labeled 'Pedestrian speed' and contains the number '0,52'.

Parameter	Value
Cell size	1
Number of pedestrians	10
Pedestrian speed	0,52

Figura 18: Campos disponibles

6.2. Cargar escenario

Para que el usuario pueda cargar un escenario a voluntad, bastará con desmarcar el check de guardar escenario tal y como se muestra en la siguiente figura.

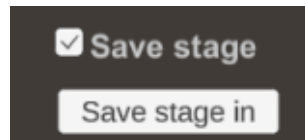


Figura 19: Cargar escenario activado

De esta manera, el botón mostrado que sirve para elegir la ruta de donde se almacenará el escenario en caso de querer guardarlo se cambiará por el botón mostrado en la figura 19, el cual abre un explorador de archivos como el de la figura 20 que se usa para elegir al archivo json que contenga el escenario, y una vez elegido el usuario podrá confirmar si se ha elegido correctamente observando la ruta del archivo mostrada tal y como en la figura 21.

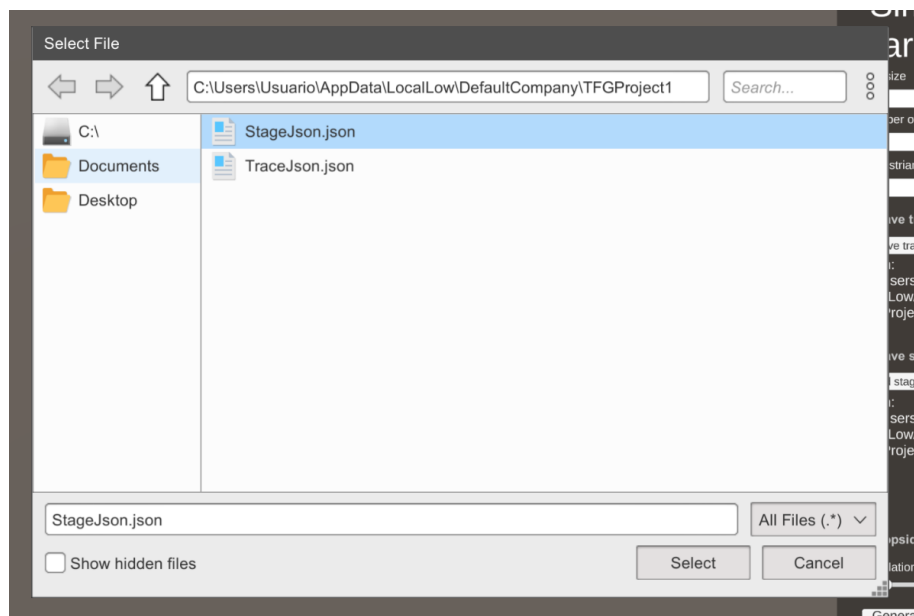


Figura 20: Explorador de archivos para elegir el escenario

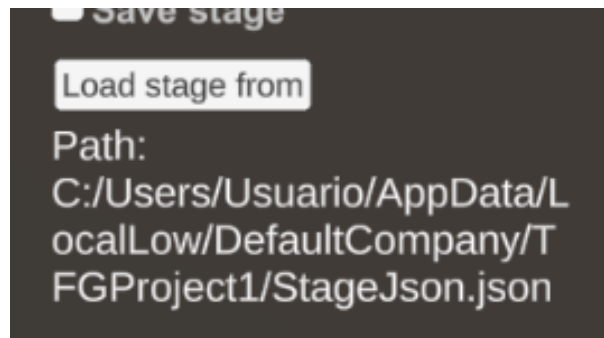


Figura 21: Ruta de escenario guardada

6.3. Guardar escenario

Al igual que para cargar un escenario elegido, para guardar el escenario que se haya usado en la simulación, bastará con tener marcado antes de iniciar la simulación el check de guardar escenario mostrado en la siguiente figura.

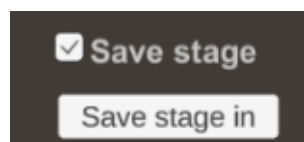


Figura 22: Guardar escenario activado

El botón mostrado que sirve para elegir el escenario a cargar se cambiará por el botón mostrado en la figura 22, el cual abre un explorador de archivos como el de la figura 23 que se usa para elegir la ruta donde se quiera guardar el archivo json que contenga el escenario, y una vez elegida la ruta el usuario podrá confirmar si se ha elegido correctamente observando la ruta del archivo mostrada tal y como en la figura 24.

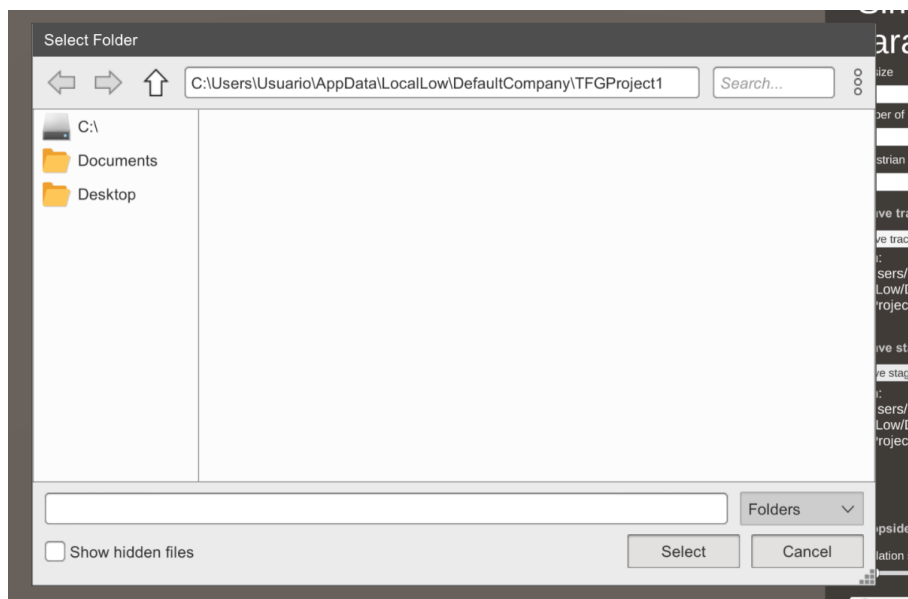


Figura 23: Explorador de archivos para elegir la ruta para el escenario

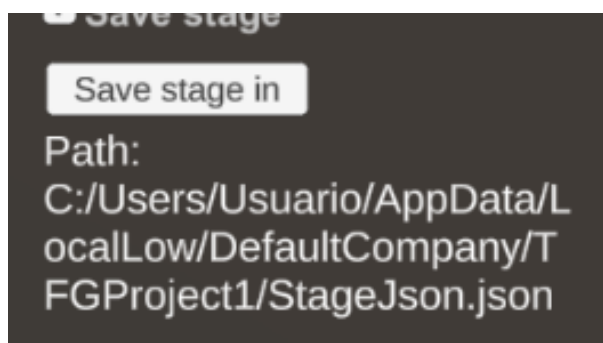


Figura 24: Ruta para escenario guardada

6.4. Cargar traza

Además del escenario, también se puede cargar y guardar la traza, siendos los procesos idénticos. Para que el usuario pueda cargar una traza a voluntad, bastará con desmarcar el check de guardar traza tal y como se muestra en la siguiente figura.

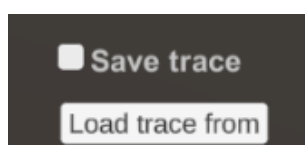


Figura 25: Cargar traza activado

De esta manera, el botón mostrado que sirve para elegir la ruta de donde se almacenará la traza en caso de querer guardarla se cambiará por el botón mostrado en la figura 25, el cual abre un explorador de archivos como el de figura 26 que se usa para elegir al archivo json que contenga la traza, y una vez elegido el usuario podrá confirmar si se ha elegido correctamente observando la ruta del archivo mostrada tal y como en la figura 27.

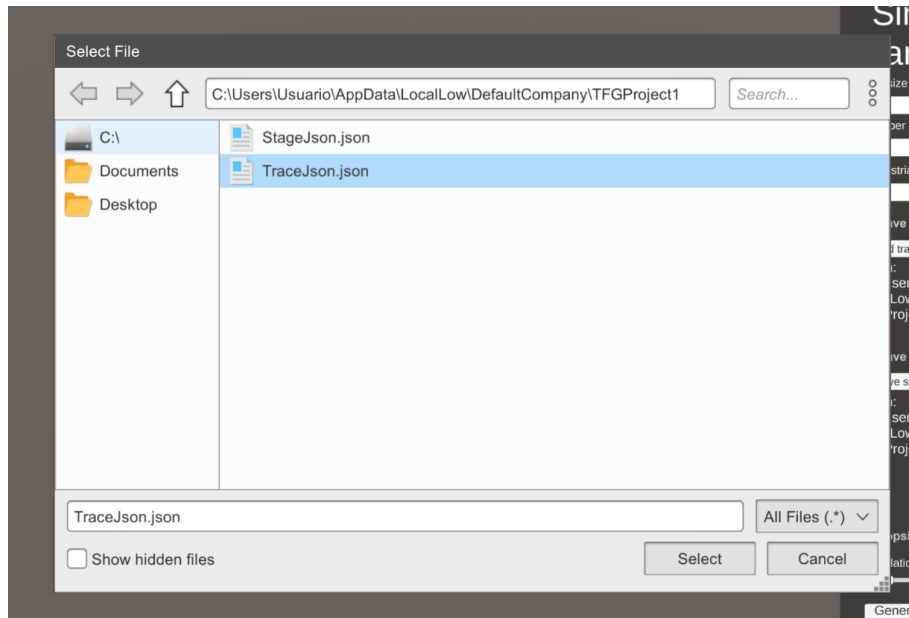


Figura 26: Explorador de archivos para elegir la traza

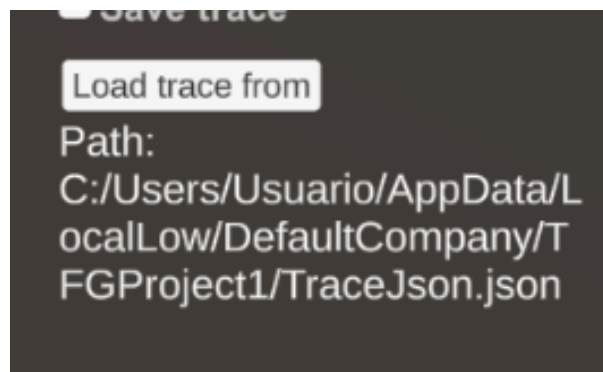


Figura 27: Ruta de traza guardada

6.5. Guardar traza

Tal y como se ha mencionado antes, la traza también se puede guardar, y para guardar la traza que se haya usado en la simulación, bastará con tener marcado antes de iniciar la

simulación el check de guardar traza mostrado en la siguiente figura.

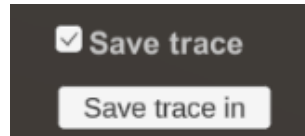


Figura 28: Guardar traza activado

El botón mostrado que sirve para elegir la traza a cargar se cambiará por el botón mostrado en la figura 28, el cual abre un explorador de archivos como el de la figura 29 que se usa para elegir la ruta donde se quiera guardar el archivo json que contenga la traza, y una vez elegida la ruta el usuario podrá confirmar si se ha elegido correctamente observando la ruta del archivo mostrada tal y como en la figura 30.

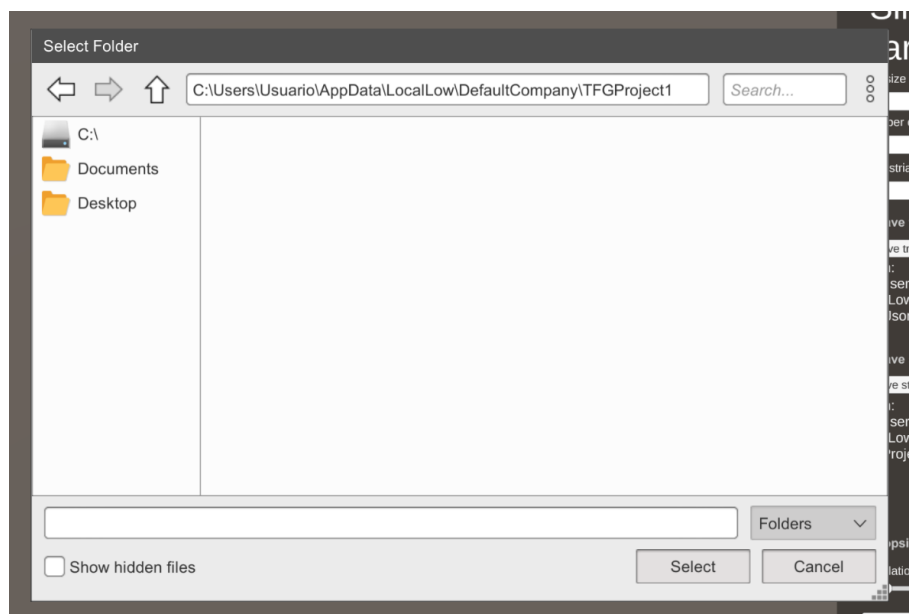


Figura 29: Explorador de archivos para elegir la ruta para la traza

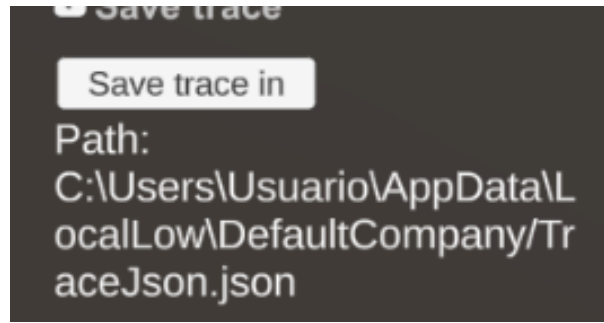


Figura 30: Ruta para traza guardada

6.6. Generar un escenario aleatorio

El usuario podrá generar un escenario aleatorio que se verá en la herramienta al momento para usarlo en la simulación. Para esto, el usuario podrá llevarlo a cabo con tan solo pulsar el botón de generar escenario que se muestra en la siguiente figura.

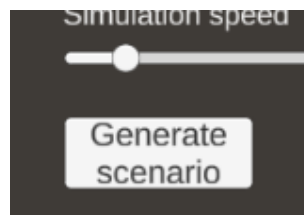


Figura 31: Botón para generar un escenario aleatorio

6.7. Iniciar simulación

Para que el usuario pueda iniciar una simulación, se deben cumplir todas las condiciones de los elementos de la interfaz para que se pueda ejecutar una simulación. Al usuario le bastará con pulsar el botón de iniciar simulación que se muestra en la figura a continuación para ejecutar una simulación.



Figura 32: Botón para iniciar una simulación

6.8. Alternar la vista

Hay dos vistas, desde arriba y libre y cada una tiene una cámara con sus respectivas limitaciones y controles. Para alternar entre estas dos vistas, basta con alternar el valor del check de cambiar vista mostrado en la siguiente figura.

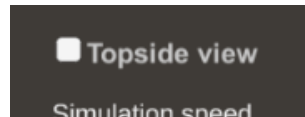


Figura 33: Casilla para alternar la vista

6.8.1. Controlar la cámara

- **Vista desde arriba:** Esta vista está controlada por una cámara que observa la simulación desde arriba con un ángulo de 90 grados. Esta cámara se mueve usando las flechas de dirección o WASD, y se puede acercar y alejar de la simulación desplazando el ratón hacia arriba y abajo respectivamente mientras se mantiene el click derecho.

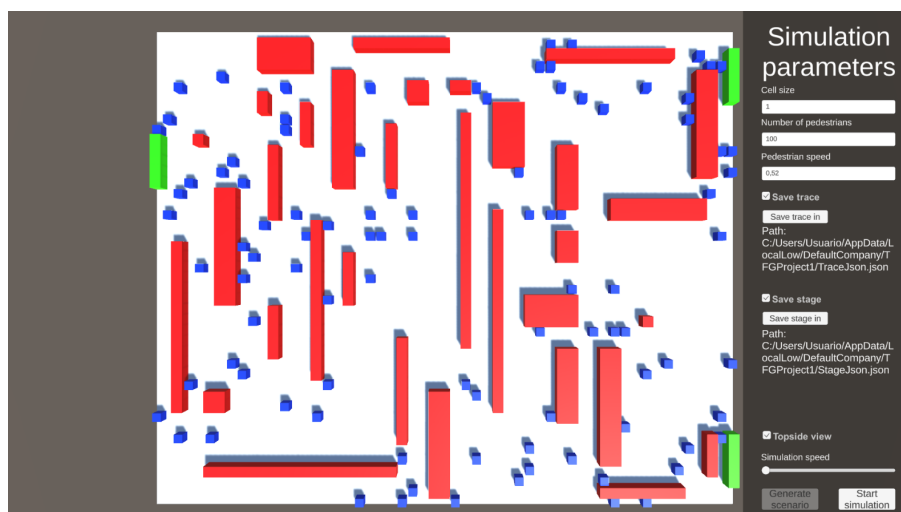


Figura 34: Vista desde arriba

- **Vista libre:** La cámara para esta vista esta en una posición inicial tal que la vista inicial en esta opción es una en picado. No obstante, esta cámara se puede mover y rotar en cualquier dirección. Al igual que la otra vista, se puede mover también con las flechas de dirección o WASD, y a diferencia de la otra vista, no tiene zoom, sino que puede

avanzar hacia delante con la W o la flecha de dirección de arriba y retroceder con la S o la flecha de dirección de abajo. Otra diferencia es que esta cámara se puede rotar con el movimiento del ratón, permitiendo a la cámara poder ver desde cualquier ángulo que se desee.

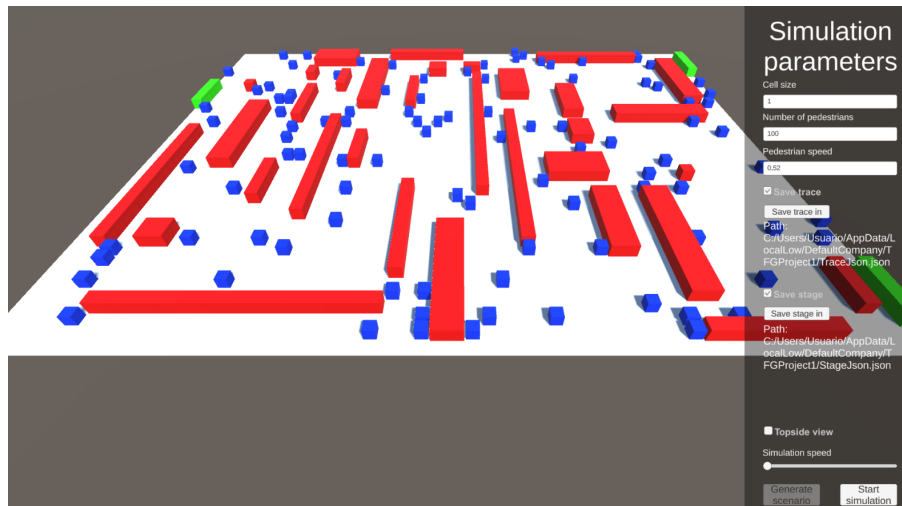


Figura 35: Vista libre

6.9. Alterar la velocidad de simulación

Una funcionalidad a tener en cuenta es que el usuario podrá cambiar la velocidad de reproducción de las simulaciones en cualquier momento, siendo su posición inicial la velocidad normal. Para esto, el usuario lo podrá hacer usando la barra deslizante que aparece en la siguiente figura.

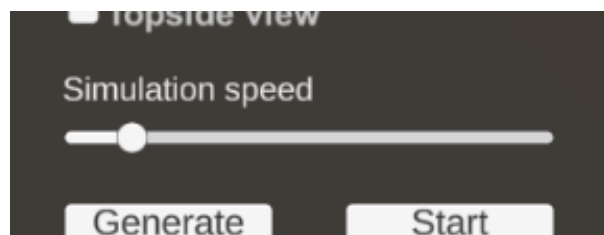


Figura 36: Barra para modificar la velocidad de las simulaciones

Experimentación

Para finalizar el proyecto, se han realizado diversos tests que muestran la efectividad que tiene el modelo surrogado y que tan alejado está del EA. Para realizar estos tests es necesario las configuraciones de cada algoritmo a comparar, los escenarios sobre los que se van a comparar, que en este contexto se les llaman instancias y la configuración que se va a aplicar a la ejecución del algoritmo. Para este proyecto, se han comparado el algoritmo evolutivo y el modelo surrogado que lo trata de sustituir sobre una única instancia.

La configuración del EA y de la simulación coinciden con la descrita en el artículo “*Evolutionary Algorithms for Optimizing Emergency Exit Placement in Indoor Environments*” [6]. En el caso de la configuración del modelo surrogado, como el modelo trata de imitar al EA, se usa la misma configuración para este.

Algunas de las más destacadas de la configuración del EA son las siguientes:

- El algoritmo se ejecuta 20 veces.
- En cada iteración se generan 99 descendientes.
- La población se inicializa de manera aleatoria.
- La Selección de padres es mediante selección por torneo binario.
- Para el reemplazo se usa el reemplazo (μ, λ) o (μ, λ) , o también conocido como μ -com. Este reemplazo consiste en escoger las μ mejores soluciones de las λ generadas por mutación y sustituirlas por el grupo anterior.

En el caso de esta configuración, el par (μ, λ) sería (100, 99), por lo que se generarían 99 soluciones además de las 100 anteriores y se intenta seleccionar 100 entre las 99, y como falta una para tener 100 soluciones seleccionadas porque es más pequeño el grupo que

lo que se debe seleccionar, se añade al grupo de 99 el resto de los mejores del grupo inicial, en este caso, 1.

Algunas de las configuraciones más destacadas de la simulación son:

- El número de simulaciones por evaluación es de 20.
- El tiempo límite es de 60s.
- La población se inicializa de manera aleatoria.
- La dimensión de las células del autómata celular es de 0.5m, y tal como se menciona en la sección 2.2.1, la vecindad es de Moore y se acompaña de Dijkstra para hallar las distancias de las celdas.
- El número de agentes es de 100.
- La velocidad base de los agentes es de 1.3m/s.
- El sesgo de atracción está en el intervalo cerrado [1.5, 2].
- La repulsión de la multitud se encuentra en el intervalo cerrado [0.25, 0.5].
- El porcentaje de velocidad aplicado respecto a la velocidad base para cada agente está en el intervalo cerrado [0.5, 1].

Una vez preparados y leídos los archivos necesarios, primero se muestran los entornos, aunque en este caso solo se ha utilizado uno llamado “low-density-1” como se puede observar en la figura 37. Tal y como se explica en el artículo “*Evolutionary Algorithms for Optimizing Emergency Exit Placement in Indoor Environments*” [6], las etiquetas “low”, “mid” y “high” acompañadas de “density” en el nombre de los escenarios utilizados indica la densidad de obstáculos, es decir, la cantidad de obstáculos que hay en proporción al tamaño de escenario. En este caso, la etiqueta completa es “low-density”, indicando que la densidad de obstáculos es baja y por tanto el número de obstáculos es bajo, estando en el intervalo cerrado [20, 30]. Como el número de obstáculos es bajo en proporción al tamaño del escenario, significa que los agentes tienen más espacio para transitar y menos probabilidades de chocarse entre ellos. Tal y como se puede observar en las figuras 37, 38, 39, 40 y 41, al final del nombre del entorno se ha añadido un 3. Este número indica el número de salidas que se van a buscar en el problema.

Después se muestran los resultados de cada algoritmo, estando dividido estos resultados en 2 gráficas, una con los mejores fitness conforme el número de evaluaciones aumenta como la que se puede ver en la figura 38, y otra mostrando la distribución de los valores de los fitness de las mejores soluciones entre todas las evaluaciones tal y como se puede observar en la figura 39.

low-density-1-3



Figura 37: Escenario base

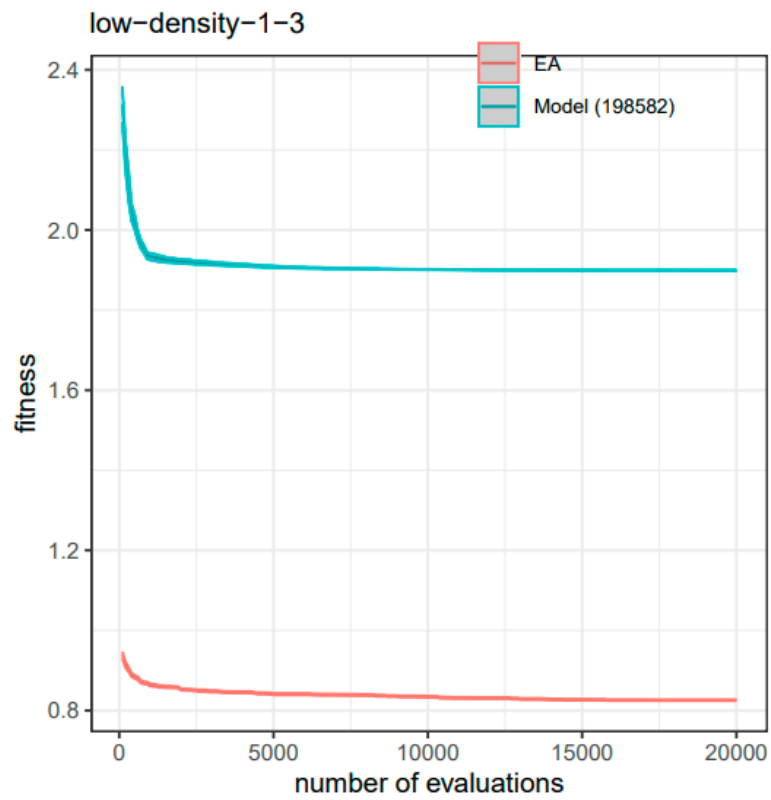


Figura 38: Los mejores fitness de cada algoritmo

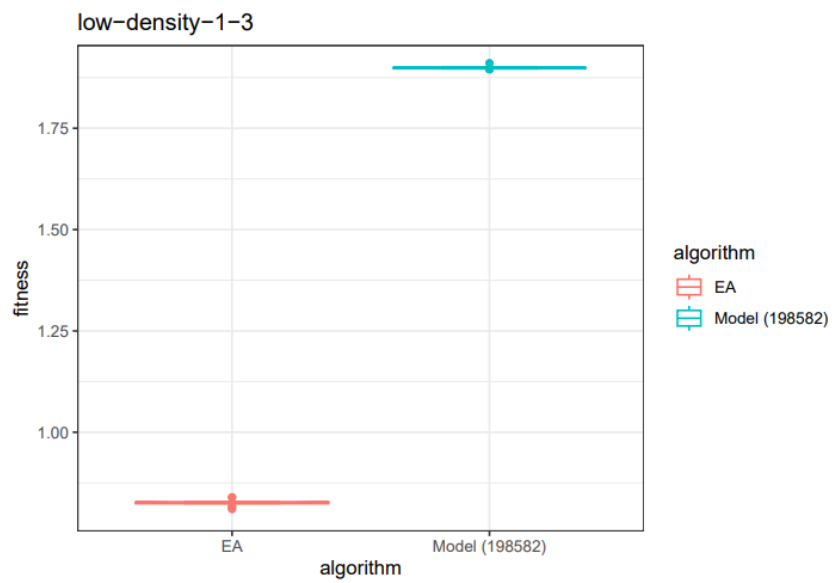


Figura 39: Los fitness de las mejores soluciones de cada algoritmo

Luego se muestra en una tabla varias estadísticas de cada algoritmo para cada instancia, como la media y la desviación típica, usándose en caso de poder algunos de los datos que se muestran en la figura 40 para los tests de Quade, de Holm con corrección de Bonferroni y Wilcoxon que se comentarán más adelante.

instance	EA				Model (198582)			
	x^*	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$		x^*	$\tilde{x}$	$\bar{x} \pm \sigma_{\bar{x}}$	
low-density-1-3	0.81	0.827	0.826 ± 0.002	★	1.895	1.899	1.899 ± 0.001	■

Figura 40: Tabla con varias estadísticas de cada algoritmo en cada instancia

Una vez mostradas las gráficas anteriores y la tabla con los datos estadísticos, se visualizan las mejores soluciones para cada algoritmo en cada instancia, por lo que en este caso solo obtendremos dos imágenes mostradas en la siguiente figura.

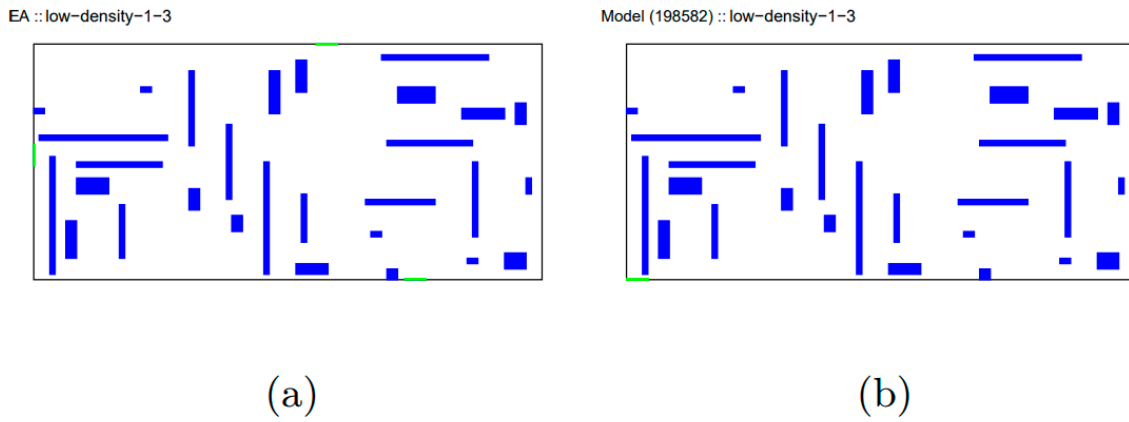


Figura 41: La mejor solución para cada algoritmo en cada instancia

El test de Quade [27] es un test para analizar múltiples bloques que asume que tiene al menos 2 tratamientos experimentales, por lo que solo se podrá ejecutar si la tabla anterior es bidimensional, es decir, si hay más de una instancia para los tests, por lo que en este caso no se ha podido realizar.

El test de Holm con corrección de Bonferroni [1], también llamado el método Holm-Bonferroni, es un test usado sobretodo para el problema de comparaciones múltiples o más conocido como Family-wise Error Rate (FWER). Como este test también necesita de múltiples instancias, no se va a poder ejecutar este test.

A pesar de que los dos tests anteriores no se hayan podido utilizar, el que sí se ha utilizado es el test de Wilcoxon [22], que a diferencia de los otros dos tests, este se utiliza para comparar muestras emparejadas o relacionadas entre si. Este test tiene hipótesis nula, la cual indica que no hay diferencias significativas entre los dos conjuntos de datos, siendo en nuestro caso entre los algoritmos.

En la tabla que se muestra en la figura 40 se puede observar un símbolo al final de cada instancia. Estos símbolos representan cual es la mejor solución por instancia, que se marca con una estrella (*), y en caso de haber diferencia significativa con los algoritmos, estos se marcan con un símbolo dependiendo de si las diferencias son estadísticamente significativas a $\alpha = 0,01(\blacksquare)$, $\alpha = 0,05(\bullet)$ y $\alpha = 0,1(\circ)$, estando en blanco en caso de no haberla. En este caso, se puede observar que en la única instancia disponible, el EA es la mejor solución, y siendo el símbolo del modelo subrogado un cuadrado negro, por lo que los resultados indican que existen diferencias significativas en todos los casos.

Tal y como se ha podido observar, los resultados no han sido muy favorables. Pienso que esto se debe a la falta de entrenamiento del modelo o/y a mi incapacidad por no poder entrenarlo correctamente puesto que no poseo el suficiente conocimiento y experiencia entrenando modelos de machine learning. Aun así, se han ejecutado los tests y en caso de continuar con este proyecto, ya se sabría que los tests se pueden ejecutar sin problemas.

8

Conclusiones y Líneas Futuras

Al ser esta la última sección, en esta se desarrolla la conclusión del proyecto y mis opiniones, además de plantear posibles líneas futuras para este proyecto.

8.1. Conclusiones

Este desarrollo surge de la idea de un proyecto de investigación, y aunque el desarrollo y la conclusión de este hayan finalizado correctamente, podría haber acabado de diferente manera. A lo largo del desarrollo, he descubierto que los proyectos de investigación no siempre finalizan con un éxito, sino que pueden surgir imprevistos, como que lo encomendado a investigar requiera más tiempo de lo previsto, y en pos de que el proyecto pueda salir adelante, habría que modificar la planificación y los objetivos de esta. Debido a esto, pienso que lo más viable en un proyecto de investigación podría ser tener unos objetivos como mínimo, y que a medida que avanza la investigación, se pueda ir modificando la planificación de esta para poder desarrollar ideas que pueden surgir a mitad del desarrollo o de una investigación o porque surgan nuevos temas a investigar.

Esto se ha visto reflejado en la planificación de este proyecto, siendo el objetivo mínimo la herramienta realizada en Unity. Casi finalizando el desarrollo de la herramienta, al grupo de investigación y a mi se nos ocurrió la idea de sustituir en un proyecto de un algoritmo evolutivo que tenía el grupo que utilizaba el autómata celular para calcular un valor numérico llamado fitness, utilizar un modelo de machine learning para esto, reduciendo drásticamente los tiempos de ejecución. Debido a que el desarrollo del proyecto iba por buen camino, decidimos intentar llevar a cabo la idea, a pesar de que la idea de sustituir el cálculo del fitness mediante el autómata celular por un modelo de machine learning requería de obtener varios

datos ejecutando el algoritmo evolutivo al menos una vez utilizando el autómata celular, además de tener que ir ajustando los parámetros del modelo en numerosas ocasiones para que los resultados obtenidos en las predicciones de este modelo fueran cada vez más acertados, por lo que costaría bastante tiempo.

Cuando antes se ha comentado que el desarrollo y conclusión de este proyecto han finalizado correctamente, es debido a que se ha cumplido el objetivo que se tenía como mínimo, que era el de la herramienta de Unity, y además de ese, se ha cumplido hasta cierto punto el de sustituir el autómata celular por un modelo de ML refinado, además de realizar tests estadísticos sobre su rendimiento. Es cierto que con más tiempo hubiese sido posible refinar el modelo para que las predicciones fuesen más acertada, pero el hecho de haber entrenado un modelo, usarlo en el algoritmo evolutivo para calcular el fitness, y realizar tests estadísticos sobre cuanto varían las soluciones y comparar los tiempos de ejecución eran la prioridad de esta parte del proyecto.

8.2. Líneas Futuras

En el futuro, debido a que es un proyecto de investigación, suelen surgir nuevas ideas a lo largo del desarrollo de este. Estas ideas suelen ser ciertas funcionalidades y aspectos que debido a razones como la escasez de tiempo no se han podido desarrollar, siendo estas ideas guardadas como posibles líneas futuras del proyecto. Algunas líneas futuras de este proyecto son:

- **Continuar el desarrollo de la herramienta en Unity:** A pesar de haber cumplido los objetivos principales para la herramienta, a esta se le pueden realizar bastantes mejoras que pueden abarcar desde la optimización de la simulación para que esta sea más eficiente y fiable hasta pulir las interfaces, ya sea mejorando la UI o añadiendo más parámetros para la simulación, como por ejemplo el número de puertas, tipo de suelo (Dijkstra, etc.), o modelos de personas.
- **Desarrollar un editor de escenarios:** Una funcionalidad que se llegó a plantear si implementarla era un editor de escenarios que se pudiese abrir desde la propia herramienta que permitiese crear y modificarlos. Esta serviría para facilitar la creación y modificación de estos, ya que para crear o modificar un escenario actualmente se tiene que hacer

modificando directamente el archivo JSON en el que está almacenado. Sin embargo, no se llegó a desarrollar debido a que por su complejidad habría que dedicarle una cantidad de tiempo considerable.

- **Añadir soporte para distintos idiomas:** Tal y como el grupo de investigación propuso, el inglés es el idioma de la herramienta ya que en un futuro era posible que el grupo presentase la herramienta en conferencias. Sin embargo, se podría haber añadido soporte para otros idiomas como el español.
- **Añadir la posibilidad de elegir entre tipos de simulación de agentes:** El objetivo principal de la herramienta es que permita configurar y visualizar la simulación de escenarios reales en los cuales se simulen movimientos de personas mediante un autó-mata celular en escenarios de emergencia que afecten a la ciudadanía, y aunque la parte de configurar la simulación se cumple mediante la modificación de algunos parámetros de esta, sería interesante poder observar simulaciones que funciones usando diferentes métodos que no sean un autó-mata celular, como por ejemplo el flocking.
- **Mejorar las predicciones del modelo actual:** El segundo objetivo de este proyecto era entrenar un modelo de machine learning para sustituirlo por el autó-mata celular que se ejecuta en numerosas ocasiones en el algoritmo evolutivo para intentar optimizar el tiempo que dura este algoritmo evolutivo. Ese objetivo se cumple con creces, sin embargo, las predicciones del modelo podrían ser más acertadas, por lo que una posibilidad era seguir ajustando el modelo para que estas predicciones sean más acertadas, pero esto hubiera llevado más tiempo del que se le quería dedicar, por lo que se paró de ajustar el modelo hasta llegar al actual. Dicho esto, no sería una mala idea continuar ajustando el modelo para que fuese aún más acertado que el modelo obtenido.
- **Añadir parámetros de entrada al modelo actual:** Al igual que las predicciones del modelo obtenido podrían ser más acertadas, también se pensó en entrenar el modelo para que admitiese más parámetros de entrada además del conjunto de las posiciones de las salidas, como por ejemplo el tipo de suelo (Dijkstra, etc.). Esta idea se descartó ya que se llegó a la conclusión de que requeriría más tiempo que el dedicado para obtener el modelo actual.

Referencias

- [1] Alexander. *Holm-Bonferroni Method: Step by Step - Statistics How To*. 18 de sep. de 2016. URL: <https://www.statisticshowto.com/holm-bonferroni-method/> (visitado 06-09-2024).
- [2] applejag. *applejag/Newtonsoft.Json-for-Unity: Newtonsoft.Json (Json.NET) 10.0.3, 11.0.2, 12.0.3, & 13.0.1 for Unity IL2CPP builds, available via Unity Package Manager*. URL: <https://github.com/applejag/Newtonsoft.Json-for-Unity> (visitado 06-09-2024).
- [3] John D Blischak, Emily R Davenport y Greg Wilson. "A quick introduction to version control with Git and GitHub". En: *PLoS computational biology* 12.1 (2016), e1004668.
- [4] Francois Chollet. *Deep learning with Python*. Simon y Schuster, 2021.
- [5] Edgar F Codd. *Cellular automata*. Academic press, 2014.
- [6] Carlos Cotta y José E. Gallardo. "Evolutionary Algorithms for Optimizing Emergency Exit Placement in Indoor Environments". En: *Applications of Evolutionary Computation - 27th European Conference, EvoApplications 2024, Held as Part of EvoStar 2024, Aberystwyth, UK, April 3-5, 2024, Proceedings, Part I*. Ed. por Stephen L. Smith, João Correia y Christian Cintrano. Vol. 14634. Lecture Notes in Computer Science. Springer, 2024, págs. 194-208. DOI: [10.1007/978-3-031-56852-7_13](https://doi.org/10.1007/978-3-031-56852-7_13). URL: https://doi.org/10.1007/978-3-031-56852-7_13.
- [7] Harvey M Deitel et al. "Java: como programar". En: (2016).
- [8] dmstern. *yasirkula/UnitySimpleFileBrowser: A uGUI based runtime file browser for Unity 3D (draggable and resizable)*. 2016. URL: <https://github.com/yasirkula/UnitySimpleFileBrowser> (visitado 06-09-2024).
- [9] Antonio Gulli y Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.
- [10] John Hunter y Darren Dale. "The matplotlib user's guide". En: *Matplotlib 0.90. 0 user's guide* (2007).
- [11] Quazi Nafiul Islam. *Mastering PyCharm*. Packt Publishing Ltd, 2015.

- [12] JetBrains. *Rider: El IDE .NET multiplataforma de JetBrains*. URL: <https://www.jetbrains.com/es-es/rider/> (visitado 06-09-2024).
- [13] Helmut Kopka y Patrick W Daly. *Guide to LATEX*. Pearson Education, 2003.
- [14] Jarosław Krochmalski. *IntelliJ IDEA Essentials*. Packt Publishing Ltd, 2014.
- [15] John D Lees-Miller. “Overleaf: Scientific writing and publishing in the age of the cloud”. En: *PKP Scholarly Publishing Conference 2015*. 2015.
- [16] Universidad de Málaga. *Bio4Res – Novel Bioinspired Optimization Techniques for Resilience and Sustainability*. 2022. URL: <https://caesium.lcc.uma.es/Bio4Res/> (visitado 06-09-2024).
- [17] Universidad de Málaga. *Caesium Research Group*. URL: <https://caesium.lcc.uma.es/> (visitado 06-09-2024).
- [18] Microsoft. *IDE de Visual Studio 2022: herramienta de programación para desarrolladores de software*. URL: <https://visualstudio.microsoft.com/es/vs/> (visitado 06-09-2024).
- [19] Microsoft. *Visual Studio Code - Code Editing. Redefined*. URL: <https://code.visualstudio.com/> (visitado 06-09-2024).
- [20] Juan Antonio Mora-Castillo. “Serialización/deserialización de objetos y transmisión de datos con JSON: una revisión de la literatura”. En: *Revista Tecnología en Marcha* 29.1 (2016), págs. 118-125.
- [21] Kandethody M Ramachandran y Chris P Tsokos. *Mathematical statistics with applications in R*. Academic Press, 2020.
- [22] Bernard Rosner, Robert J Glynn y Mei-Ling T Lee. “The Wilcoxon signed rank test for paired comparisons of clustered data”. En: *Biometrics* 62.1 (2006), págs. 185-192.
- [23] Ken Schwaber y Jeff Sutherland. “La guía de Scrum”. En: *Scrumguides. Org* 1 (2013), pág. 21.
- [24] José Antonio González Seco. “El lenguaje de programación C#”. En: URL: https://programacion.net/articulo/el_lenguaje_de_programacion_c_167/3. Fecha de consulta 29 (2001).
- [25] Nishant Shukla y Kenneth Fricklas. “Machine learning with TensorFlow”. En: (2018).

- [26] Swati Singh y Amanpreet Kaur. “Game development using unity game engine”. En: *2022 3rd International Conference on Computing, Analytics and Networks (ICAN)*. IEEE. 2022, págs. 1-6.
- [27] National Institute of Standards y Technology. *QUADE TEST - Information Technology Laboratory*. URL: <https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/quade.htm> (visitado 06-09-2024).
- [28] Unity. *About Unity Test Framework | Test Framework | 1.1.33*. 14 de nov. de 2023. URL: <https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html> (visitado 06-09-2024).
- [29] Stefan Van Der Walt, S Chris Colbert y Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. En: *Computing in science & engineering* 13.2 (2011), págs. 22-30.
- [30] John Verzani. *Getting started with RStudio*. “Reilly Media, Inc.”, 2011.
- [31] Vivi Nur Wijayaningrum, Vivin Ayu Lestari et al. “Jupyter lab platform-based interactive learning”. En: *2022 International Conference on Electrical and Information Technology (IEIT)*. IEEE. 2022, págs. 295-301.
- [32] Yihui Xie. “TinyTeX: A lightweight, cross-platform, and easy-to-maintain LaTeX distribution based on TeX Live”. En: *TUGboat* 40.1 (2019), págs. 30-32.
- [33] Zhi-Hua Zhou. *Machine learning*. Springer nature, 2021.

Apéndice A

Manual de Instalación

En este apartado, se indicará cómo se puede instalar y ejecutar la herramienta. La herramienta por el momento solo se puede usar a través de un ejecutable en local en Windows mediante los siguientes pasos:

1. Descargar el archivo comprimido nombrado como “SimulationTool.zip” desde la release que se llama “[Simulation tool build](#)” que se ubica en el apartado de “Releases” en el [repositorio](#) en GitHub.
2. Descomprimir el archivo descargado, el cual contiene una carpeta llamada “Simulation-Tool” con el ejecutable de la herramienta de simulación.
3. Lanzar el ejecutable de la herramienta “TFGProject1.exe” dentro de la carpeta descomprimida.



UNIVERSIDAD
DE MÁLAGA

| **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga