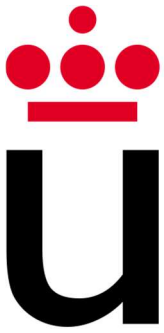

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Diseño y Desarrollo de Videojuegos
Curso 2022-2023

Trabajo fin de Grado

VISUALIZACIÓN DE ALGORITMOS DE BÚSQUEDA PARA RESOLUCIÓN DE LABERINTOS

Autor: *Sergio Benítez Vicente*

Tutores: *Juan José Pantrigo Fernández*
Jesús Sánchez-Oro Calvo

Agradecimientos

Nunca había pensado que podría alcanzar este momento, pero al final se ha conseguido a base de esfuerzo y ganas. Siendo sanitario, con un trabajo a media jornada, al principio, y mucha ilusión por realizar este grado, se puede decir que al final todo esto ha dado sus frutos.

Tengo que agradecer esto a mis padres que, aún teniendo sus más y sus menos a lo largo de mi infancia, me han inculcado una cultura de la perseverancia y del esfuerzo que quizás no habría conseguido de otra manera. En los momentos en que todo parecía que estaba mucho más cuesta abajo, han sido los primeros que le han quitado hierro al asunto y han querido que siguiera, porque al final del todo, esto es algo que estaba haciendo por gusto no por obligación.

Una pequeña parte de todo esto se lo debo también a mis amigos que han estado apoyándome, sobre todo en mi primer año de carrera donde mi mayor apoyo no estaba y ellos estuvieron animándome a dar el 100% de mí para que esto saliera adelante, gracias a mi pasión por los videojuegos. Desde animarme en la lejanía, hasta acompañarme en algunas aventuras, han conseguido que lo viera como algo más y me han hecho sentir realmente bien.

Aunque estos agradecimientos están bien, ninguno puede compararse al de mi mayor apoyo en todo esto: Laura. Ella ha estado ahí en todos los buenos y los malos momentos, siendo, realmente, la culpable de que esté terminando este Grado en Diseño y Desarrollo de Videojuegos. Después de 3 años tras graduarme como Enfermero, ella sabía que no estaba del todo contento con lo que hacía y me animó a que probara suerte con este grado, siendo la chispa que encendió todo.

Ella siempre me suele decir que todo esto no hubiera sido suficiente si no me hubiera esforzado y que todo ha dependido de mí, pero si no fuera por sus palabras en ningún momento habría encontrado tanto placer en realizar este tipo de trabajo como lo que yo pensaba. He descubierto que programar y realizar videojuegos es algo con lo que disfruto y si no hubiera insistido en que lo hiciera, me habría perdido todo esto.

¡Muchas gracias a todos ellos y espero poder seguir cumpliendo este sueño!

Resumen

Un elemento importante en la elaboración de los videojuegos son los algoritmos en el ámbito de la programación. Conocer la forma de poder resolver problemas de la manera más eficiente es una tarea muy importante para un programador. A esto hay que añadir que conocer la teoría de los mismos puede llegar a ser fácil de entender en una primera instancia, pero para poder implementarlo suele generarse algo más de dificultad.

A través de este TFG se va a unificar la parte teórica de diversos algoritmos de programación con una visualización de lo que realiza en cada momento. De esta forma, se busca elaborar una metodología de trabajo por la que un solo individuo sea capaz de establecer un entorno y unas herramientas para crear una aplicación interactiva para un público tan amplio como el de los alumnos del Grado en Diseño y Desarrollo de Videojuegos.

Con las herramientas escogidas se elabora una interfaz inicial de usuario y un entorno para aplicar los algoritmos de búsqueda: un laberinto. Dicho entorno es elaborado de forma aleatoria con el fin de englobar muchas más opciones. Una vez realizado se escogen los métodos de búsqueda para resolver dicho entorno y se aplica su visualización, unificando un entorno de imágenes con código que sirva para entender el algoritmo que se explora.

Juntando todo esto acaba formándose una aplicación interactiva útil para poder estudiar diferentes algoritmos de búsqueda, tanto en grafos como fuera de ellos, permitiendo un gran margen de mejora, con posibles actualizaciones futuras para más tipos de algoritmos.

Palabras claves:

- Algoritmos para videojuegos
- Algoritmos de búsqueda
- Resolución de algoritmos
- Visualización de algoritmos

Índice de contenidos

Índice de imágenes

1

INTRODUCCIÓN

Uno de los aspectos que más suele abrumar a todo estudiante en el ámbito de la ingeniería suele ser el desarrollo técnico de aplicaciones. Es importante conocer los lenguajes de programación que se van a utilizar, según el ámbito donde cada individuo se encuentre, pero dentro de ellos existe una serie de herramientas que pueden hacer el trabajo más fácil para la resolución de algunos problemas: los algoritmos de programación.

El ámbito de los videojuegos no iba a ser diferente, ya que recoge el testigo del desarrollo software en ámbitos como el de las aplicaciones de escritorio o aplicaciones web. De esta forma, todo lo que tiene que ver con la parte técnica de los videojuegos se ha ido adueñando de ciertos paradigmas o algoritmos del entorno de la programación. Analizarlos y entenderlos forma parte de cualquier programador que trabaje en ellos, por lo que toda herramienta que ayude a ello será bien recibida.

Algo muy relacionado con los algoritmos son las estructuras de datos, otro elemento relevante en el desarrollo de software y que suele ser fácil de explicar, en una primera instancia, pero difícil de consolidar en la mente de un programador. En este último punto llega la principal motivación de este TFG, ya que se busca poder visualizar todo lo que conlleva un algoritmo de programación para que un estudiante sea capaz de comprenderlo mejor.

Si se llegan a juntar estos dos conceptos relacionados con el apartado técnico del desarrollo software, puede ser algo bastante complejo conseguir entenderlo para un estudiante que acaba de empezar con ello. El nivel de abstracción que se pide cuando se realiza la programación de estructuras de datos y de algoritmos genera una importante barrera en cualquier estudiante novel.

Con el paso del tiempo se ha ido observando que usar herramientas de visualización para entender mejor los conceptos de programación ha sido una gran idea. En la actualidad se

observa siempre a la hora de depurar código, que es más favorable tener un vistazo rápido según avanza el código, para localizar el error al momento, que tratar de adivinarlo revisando todo el código en sí. Para conceptos como los anteriormente mencionados se pueden vislumbrar algunas herramientas ya existentes: [1]

- ***Graph Playground:*** herramienta online útil para el correcto estudio de los grafos y las diferentes formas de manipularlos. Sin presentar una estética tan atractiva como otros, sirve como una perfecta carta de presentación para este tipo de estructura y las diferentes operaciones que se pueden hacer con ella.
- ***Visualgo.net:*** aplicación online bastante completa que trata de englobar en 24 módulos diferentes tipos de estructuras de datos con algoritmos de manipulación de datos dentro de ellas. Todo queda mostrado con un aspecto visual muy llamativo, que junta un diseño esquemático con parte del código que se va realizando en cada interacción.

Partiendo de este tipo de herramientas que se pueden ver de forma online, se ha querido ser más específicos para este TFG, centrando los esfuerzos en algo más conocido en el ámbito de los videojuegos. Sin lugar a duda, los mapas y el desplazamiento por ellos es bastante habitual en cualquier experiencia interactiva, por lo que el objetivo principal era poder explorar métodos de resolución de laberintos.

Dentro de los videojuegos existen diferentes algoritmos que colaboran con esto, siendo los más usados aquellos que parten de grafos. Llegar a plasmar visualmente cualquier laberinto y darle una solución ha sido la principal motivación para este TFG y analizando varios de los algoritmos estudiados a lo largo del grado se ha buscado unificar todo en un mismo lugar.

Sin alejarse mucho de la parte más didáctica y usando algunos trabajos de alumnos anteriores del grado, se busca poder hacer visible estos algoritmos de resolución de laberintos combinando tanto la parte más técnica del código como la navegación interactiva a través del entorno que se intenta explorar. De esta forma, cualquier estudiante que quiera estudiar estos conceptos de estructuras de datos o de algoritmos de búsqueda podrá recurrir a él.

Realizar toda esta tarea en un entorno conocido por los alumnos del grado y dejando que puedan navegar libremente en un laberinto que trata de resolverse, supondrá un avance para el estudio y comprensión de los grafos y los métodos de búsqueda. Englobarlo todo

en un entorno que pueden llegar a conocer y que genera un mayor atractivo, emularlo con un aspecto de videojuego en un entorno usado para ello motivará a muchos más alumnos a realizar trabajos similares y a comprender mejor todo este punto de teoría que puede resultar tedioso.

2

OBJETIVOS

- ***Objetivos Generales:***
  - Programación de diferentes algoritmos de búsqueda para resolver la salida de laberintos.
  - Visualización de la resolución de algoritmos de búsqueda para resolución de laberintos con imágenes y código.

- ***Objetivos Específicos***
  - Planificación de proyecto para visualizar algoritmos de búsqueda para resolución de laberintos.
  - Programación de creación aleatoria de laberintos.
  - Selección de algoritmos de búsqueda para resolución de laberintos.
  - Implementación de algoritmos de búsqueda para resolución de laberintos.
  - Visualización en imágenes y código de los diferentes algoritmos de búsqueda para resolución de laberintos.
  - Diseño de la estética para toda la aplicación.

3

METODOLOGÍA

La incapacidad de realizar este tipo de trabajo en equipo ha supuesto que un único individuo sea el capacitado para todos los aspectos del desarrollo de software, limitando mucho la elección de la metodología. Esto ha supuesto el descarte de varias metodologías ágiles, ya que requieren de la definición exacta de los individuos en cada apartado, tratando de ir más hacia una metodología tradicional donde todo queda mucho más generalizado. Aunque se trata de una metodología mucho más pesada y laboriosa, para un único individuo, ha resultado satisfactoria y mucho más llevadera. [2]

Englobando todas las posibles metodologías tradicionales, y teniendo en mente el modo de trabajo con el que estar más cómodo, se ha decidido por una metodología de software llamada incremental. Para este tipo de metodología se busca ir creando un proyecto desde cero con etapas que van aumentando funcionalidades en cada iteración en la que se va mejorando y elaborando la aplicación. [2]

Usando este tipo de metodología se tiene como cliente de la aplicación a los tutores del TFG, siendo la forma de poder ir elaborando nuevas iteraciones en el desarrollo del software revisando los resultados de cada una de ellas. De esta forma, cada iteración en la que se comience a trabajar presentará cuatro etapas que se repiten en cada una de ellas, siempre con toda la información recopilada con el supuesto cliente.

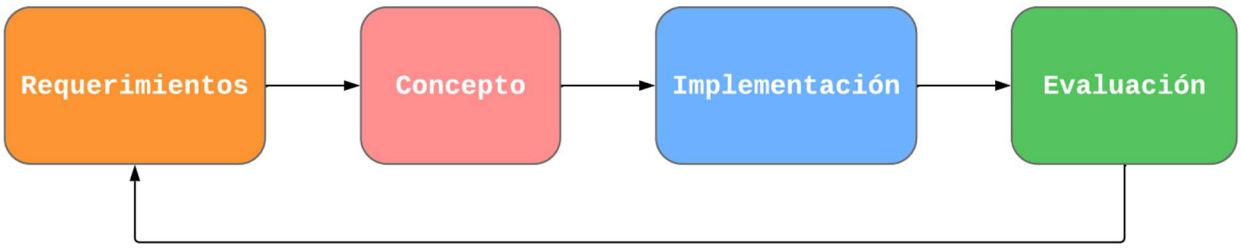

Ilustración 1 - Flujo metodología incremental

- Elaboración de requerimientos de cada iteración.

- Diseño conceptual de las modificaciones que tendrá la aplicación.
- Implementación de las modificaciones que añaden los requerimientos.
- Evaluación del resultado final.

Con la última de estas etapas de la metodología se recoge la información necesaria para volver a empezar, en caso de que no se haya terminado el desarrollo del software al completo. Gracias a ello se obtiene una retroalimentación continua con las evaluaciones y permite acomodar los cambios de forma progresiva, marcando siempre pequeños hitos que se van expandiendo.

El hecho de haber sido realizado el trabajo por una persona a lo largo de un único año lectivo ha permitido dedicar el tiempo que se podía cada día trabajado, de forma que no siempre se le dedicaba el mismo. No obstante, el trabajo ha quedado dividido en diversas iteraciones que son determinadas por los requerimientos obtenidos y cada una de ellas ha tenido un tiempo orientativo de duración. A través del siguiente diagrama de Gantt, dividido en semanas, se puede comprobar el tiempo destinado a cada tarea.

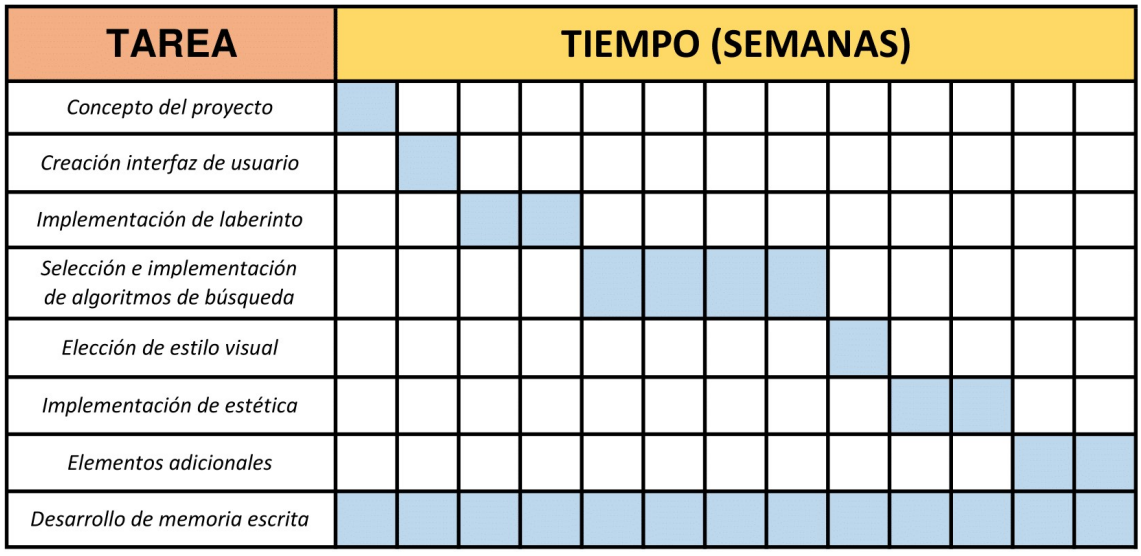

Ilustración 2 - Diagrama de Gantt del proceso

Cada uno de estos apartados pueden ser susceptibles a pequeños cambios de tiempo en su ejecución, siempre dependiendo de la dificultad de los mismos. No obstante, y siguiendo la principal característica de esta metodología tradicional, en todo momento se trabajará con el proyecto al completo, tratando de cumplir los requerimientos que se obtienen tras cada revisión de la aplicación. Una a una se irá analizando cada etapa en la que se ha desglosado el trabajo fin de grado, indagando más en los puntos más técnicos del desarrollo de software.

4

DESCRIPCIÓN INFORMÁTICA

4.1- SELECCIÓN DE ENTORNO Y HERRAMIENTAS

Analizando el proyecto en cuestión, es importante escoger bien las herramientas que se encargarán de la parte técnica y de la parte artística, con el fin de ser directo y fácil para su posterior uso por alumnos. De esta manera, se han escogido las siguientes herramientas para la elaboración del TFG.

- ***Unity 2021***: siendo uno de los motores de videojuegos más tratados a lo largo del grado universitario y presentar bastante facilidad para la elaboración de entornos de interfaces, ha sido considerada la herramienta perfecta para llevar a cabo esta aplicación. Basado en scripts codificados en el lenguaje de C# y mostrar un entorno muy intuitivo con objetos y jerarquías dentro del propio programa, es considerada una herramienta muy útil para el desarrollo de aplicaciones que requieran de interacción con cualquier tipo de usuario. [3]

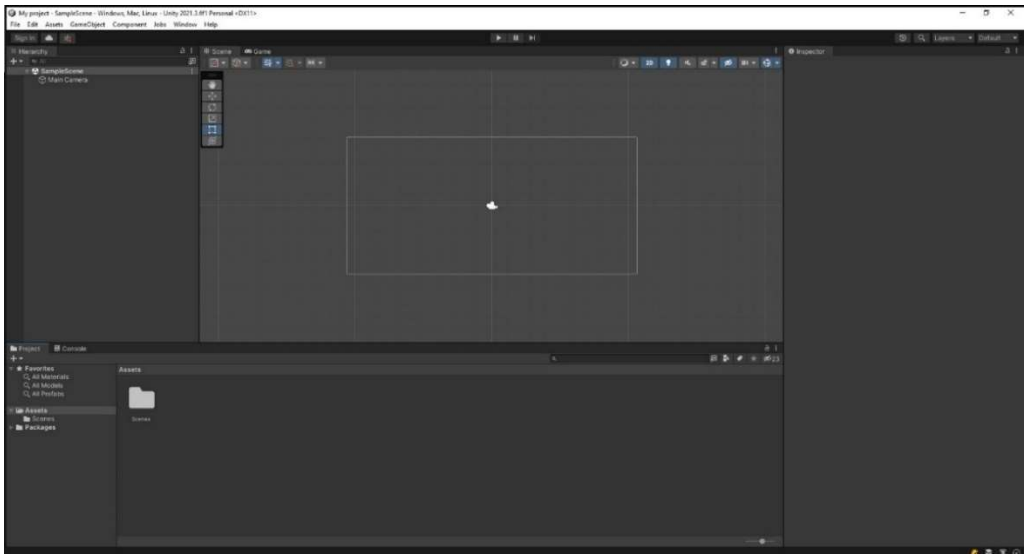

Ilustración 3 - Imagen de entorno de Unity 2021

Existen diversas herramientas para poder generar aplicaciones de este estilo, como puede ser Unreal o Game Maker, ambas tratadas en el grado. No obstante, se ha decidido escoger Unity por tratarse de la más accesible dentro de varias asignaturas y la más conocida para el entorno en 2D que se trata de buscar. Observando que va a ser una aplicación que puede ayudar a otros alumnos del grado, usar una herramienta que usan de forma más común a lo largo de las asignaturas puede servir de motivación para que quieran seguir explorándola, mostrando todo lo que puede llegar a plantear.

- ***Adobe Photoshop 2020:*** ya que se va a tratar un entorno 2D, se ha decidido escoger la herramienta por excelencia para poder crear y manipular imágenes, como es el caso de Adobe Photoshop. Al igual que ocurría con Unity, se trata de la más utilizada y de la que mayor información tiene, por lo que su uso resulta muy sencillo.

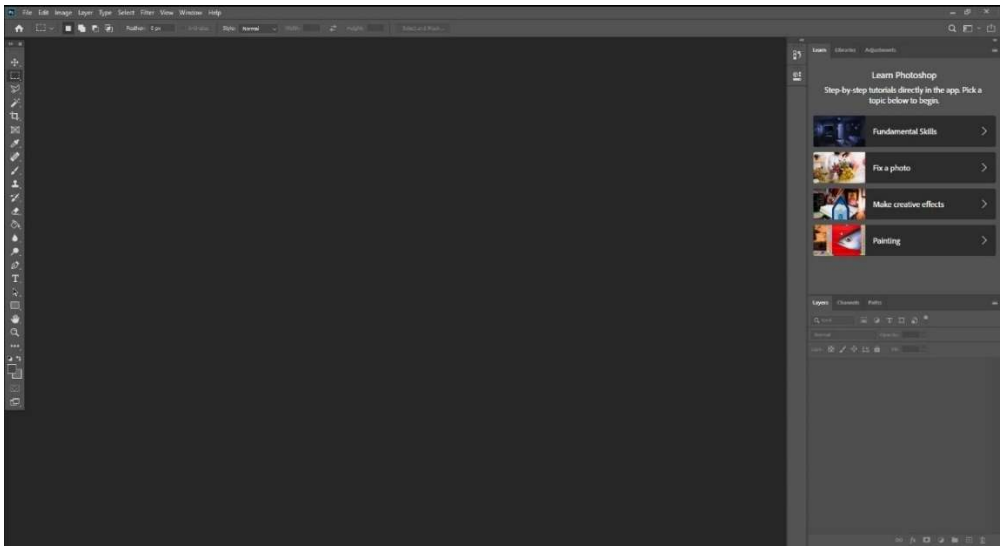

Ilustración 4 - Imagen de entorno de Adobe Photoshop 2020

Ha sido escogida la versión de 2020, ya que contenía todas las herramientas necesarias para poder trabajar en la aplicación, sin necesidad de ampliar su gran catálogo con ediciones más modernas. Las imágenes utilizadas en el proyecto son de elaboración propia o de repositorios de libre distribución.

4.2- ELABORACIÓN DE DIAGRAMA DE FLUJO DE APLICACIÓN

Buscando la simplicidad y el propósito didáctico, la aplicación ha sido planteada como una navegación continua entre interfaces de usuario. Por esta última razón, cada una de las interfaces va a tener una función clara, pero tampoco se deben de sobrecargar con exceso de información al usuario, permitiendo un uso intuitivo de la misma.

Un punto a favor de este tipo de proyecto es que su uso va a estar destinado a personas familiarizadas con el uso de PC y la información que se va a tratar, contando con un manual de instrucciones previo que lo refuerza. De esta manera, se puede llegar a obviar el mero hecho de elaborar algún tipo de tutorial, como si de un videojuego se tratase. Observando todos estos puntos, la aplicación ha quedado reducida a tres tipos de pantallas que verán los usuarios:

Ilustración 5 - Diagrama de flujo

- ***Pantalla de Título:*** esta pantalla es la utilizada para dar introducción a todo el proyecto. Sirve para mostrar el título de la aplicación. Dispone de un botón que hace que el usuario vaya directamente hacia la pantalla de selección.
- ***Pantalla de Selección:*** aquí se elige la complejidad y el algoritmo a estudiar. Presenta un único botón para acceder a la pantalla de visualización, además de los desplegables donde elegirá una opción u otra.
- ***Pantalla de Visualización:*** es la interfaz donde se genera el laberinto de forma aleatoria y se ejecuta la resolución del mismo. La primera vez que se observa se pueden ver 2 botones, uno para inicializarlo todo y otro para volver a la pantalla de selección. Una vez que se presiona el botón de inicio aparecerá la zona donde se va a trabajar, centrada en un tablero que sirve como laberinto, unos botones para avanzar por la resolución y un espacio donde estará escrito el código de cada paso que se dé. No obstante, dentro del propio laberinto va ocurriendo una animación que permitirá ver cómo el algoritmo intenta resolverlo todo. Desde aquí se podrá volver a la pantalla de selección en cualquier momento, siendo el único modo de salir de esta interfaz, a no ser que acabe la ejecución del laberinto.

Con el fin de poder cerrar la aplicación en el momento que el usuario quiera, se ha elaborado de forma que no englobe todo el tamaño de la pantalla, así cuenta con el botón de salida habitual para este tipo de situaciones.

4.3- PRIMERA VISUALIZACIÓN DEL INTERFAZ DE USUARIO

La estructura básica para la aplicación hace que deba contener tan solo una cámara, un sistema de eventos y un canvas con las interfaces que se van a mostrar al usuario. Debido a la forma de trabajar en Unity, un primer contacto va a ir destinado a dedicar parte del tiempo inicial a establecer correctamente la estructura del proyecto. Por todo esto, la escena principal en Unity estará dividida en objetos vacíos que conforman cada una de

las interfaces de la aplicación, conteniendo en su interior los diferentes elementos predefinidos en Unity para funciones de interfaz. [4]

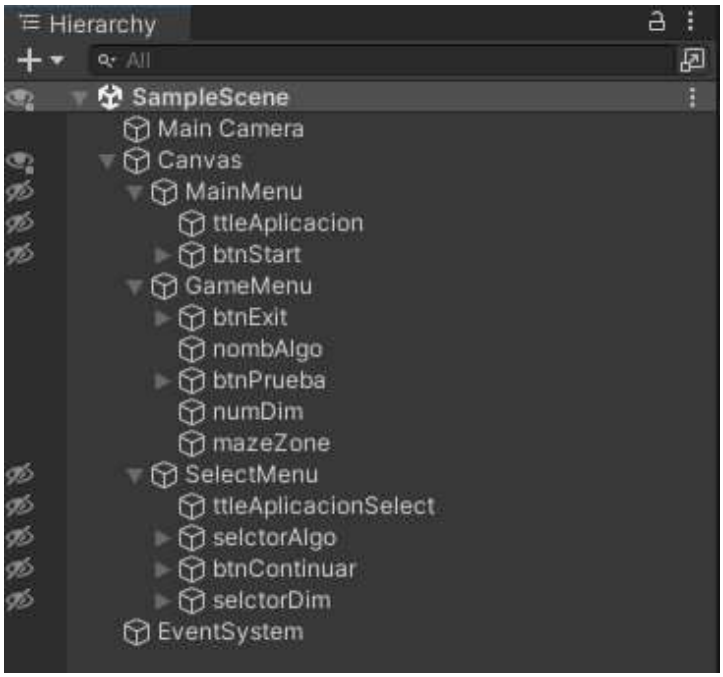

Ilustración 6 - Jerarquía de objetos de interfaces

Tras establecer la estructura de todo el proyecto se realiza un primer vistazo a todos los elementos y se programan los diferentes eventos de navegación. Se tratan de elementos establecidos por el entorno como elementos de interfaces de usuario, de forma que algunos están previamente programados para que el desarrollador pueda llegar a estar en una capa más externa del código.

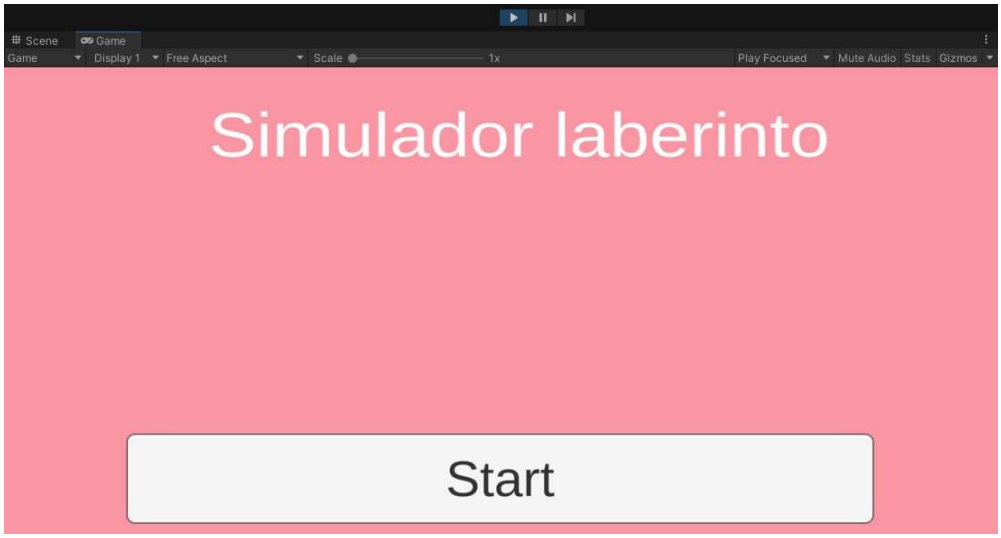

Ilustración 7 - Pantalla de Título en placeholder

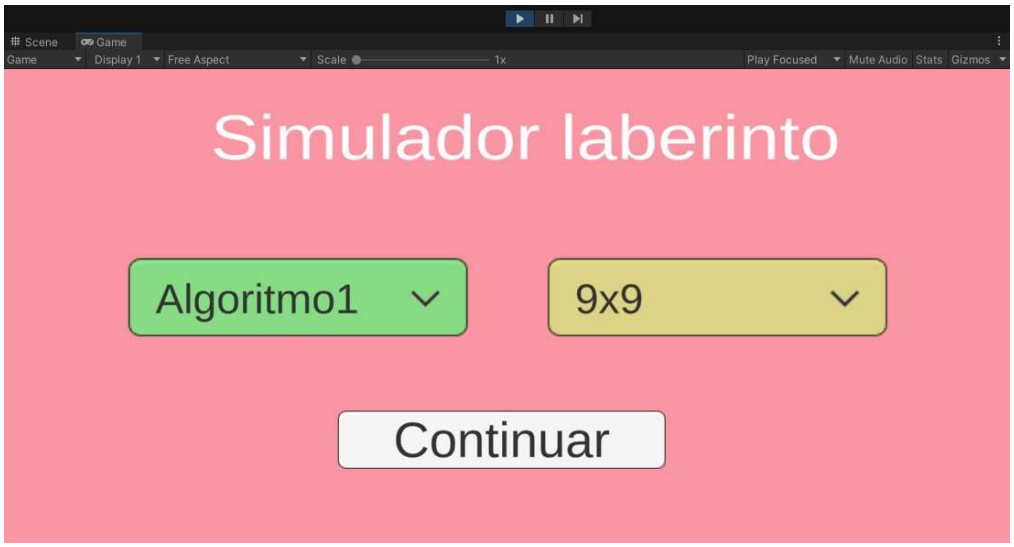

Ilustración 8 - Pantalla de Selección en placeholder

Ilustración 9 - Pantalla de Visualización en placeholder

Aunque los elementos visuales son sencillos, el objetivo era realizar la estructura interna del código por lo que se han establecido dos *scripts* para la visualización de las interfaces. Dentro de estos *scripts* se ha conseguido combinar los elementos visuales de Unity y las diferentes funcionalidades, cada uno de ellos con el siguiente contenido:

- ***GameInterfaceControls:*** este *script* implementa la navegación por la interfaz. Contiene todas las funcionalidades de botones y qué elementos deben de estar visibles. Está asociado al objeto Canvas, el cual recoge en su interior todos los GameObjects relacionados con interfaces de usuario.
- ***GameFunctions:*** realiza todos los cálculos y modificaciones de los elementos vistos por pantalla en la interfaz de visualización del laberinto. Contiene todos los métodos que pueden llegar a ser llamados dentro de la visualización. Este script

está asociado al objeto que recoge en su interior toda la pantalla de visualización, además de todos los elementos con los que se interacciona.

En una primera instancia, y para controlar tamaños de la aplicación, se ha establecido una zona de pantalla para el laberinto. Para representar la zona de resolución del algoritmo se usa un tablero de tamaño NxN. Esta zona va a quedar dividida en celdas cuadradas que representan cada sala del laberinto y sus conexiones entre sí.

Toda esta sección ha sido codificada como un array de un nuevo objeto que no está en Unity y que ha sido creado únicamente para esta aplicación llamado "celda". De esta forma, se consigue observar una cuadrícula que representa al laberinto y sirve como un primer vistazo a lo que podría crearse más adelante, como si se tratase de una vista cenital.

Ilustración 10 - Ejemplo básico de zona de laberinto

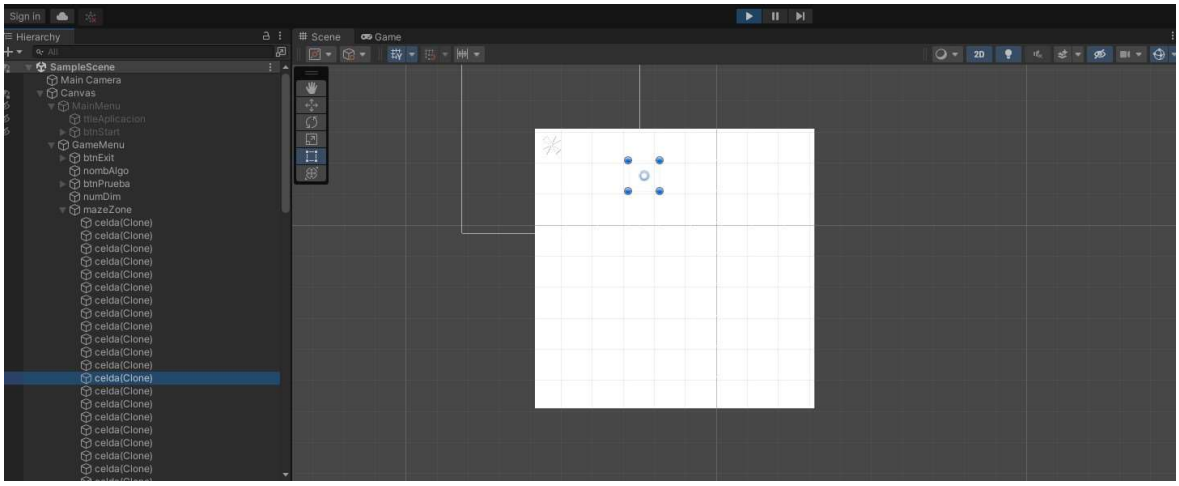

Ilustración 11 - Vista de edición de zona de laberinto

Al final del todo se puede llegar a observar una cuadrícula de celdas que conforman el laberinto. Cada una de estas celdas será un objeto propio en Unity que proporcionará información acerca de los muros que va a tener, pero esto será utilizado cuando se quiera

crear el laberinto de forma aleatoria. Lo que se ha establecido aquí es una zona de trabajo donde se muestra el laberinto, además de codificar el tamaño que puede tener cada cuadrícula y la distancia entre las casillas.

4.4- CREACIÓN ALEATORIA DEL LABERINTO

Para empezar, hay que crear las cuadrículas que van a conformar el laberinto desde el código. Se realiza a través de un *script* que va a determinar el comportamiento y las características de cada una de las celdas que conforman el laberinto, con variables para codificar el estado de sus paredes, su posición y su estado de ocupación.

La codificación de este algoritmo requiere la creación de una clase nueva llamada Cell, la cual va a disponer de todas las variables necesarias para conocer las conexiones entre las celdas del laberinto. Dentro del *script* de GameFunctions, se ha usado una matriz de objetos Cell y se ha definido el laberinto para más adelante rellenarlo.

Para crear el laberinto se escoge una casilla aleatoria y se comienza a crear todo lo demás a partir de ella. Usando un algoritmo de creación de laberinto con base en el *backtracking*, se seguirá el siguiente esquema para poder elaborar cualquier tipo de laberinto: [7]

1º Se crea una matriz de NxN celdas y se aplican los muros que no van a poder modificarse de los bordes.

2º Cada una de estas celdas va a tener una lista interna con las posibles direcciones que va a tomar, ordenadas de forma aleatoria.

3º Se escoge una celda al azar para comenzar a crear un camino y se marca como celda visitada.

4º Se recoge la próxima dirección que se va a tomar del listado que se ha rellenado previamente en cada una de las celdas. Si la celda que se va a visitar no está todavía visitada se accederá a ella, mientras que en el caso de que lo esté se escoge otra dirección.

5º Para cada momento que se accede a alguna celda se modificarán los muros que hay entre esas dos, de forma que todas empiezan con todos los muros colocados hasta que haya un camino entre ellas.

6º El algoritmo de creación de laberintos terminará cuando se hayan conseguido visitar todas las casillas que conforman el laberinto, de forma que habrá un camino que pueda unirlas todas.

Para las primeras pruebas de creación de laberinto se han elaborado *placeholders* simples de cada posible caso para las celdas. Se han elaborado un total de 16 imágenes según los diferentes resultados, teniendo en mente las cuatro direcciones y las diferentes combinaciones entre ellas. Algunos ejemplos de laberintos elaborados con el algoritmo escogido se muestran en las ilustraciones 12 y 13.

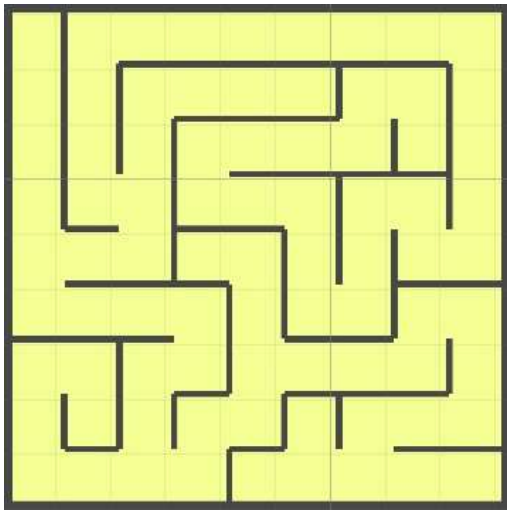

Ilustración 12 – Ejemplo 1 laberinto creado de dimensión 18

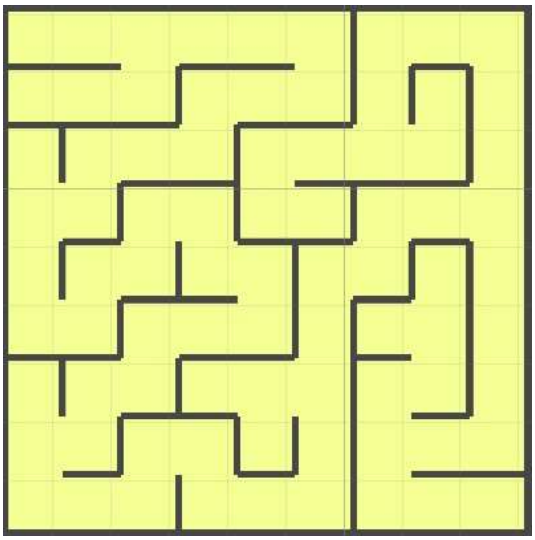

Ilustración 13 - Ejemplo 2 laberinto creado de dimensión 18

Las dimensiones escogidas para la elaboración del laberinto pueden modificarse en cualquier momento. Se han implementado 3 posibles tamaños para el laberinto, algunos más útiles para la depuración y otros más útiles para la visualización. Empezando con un

tamaño de 3x3, se ha decidido ampliar a tamaños múltiplos, como han sido 9x9 y 18x18 con el fin de situar al algoritmo en escenarios más complicados.

4.5- SELECCIÓN DE ALGORITMOS DE RESOLUCIÓN DE LABERINTOS

Con el fin de proporcionar diferentes herramientas a los usuarios se han implementado tres métodos de resolución de laberintos. Cada uno utiliza un tipo de razonamiento diferente para salir del laberinto, de forma que el usuario pueda comprender cada una de las formas en las que se puede resolver este problema. Todos ellos han sido utilizados en alguna ocasión dentro del grado, analizándose de forma individual con apoyo visual para esta aplicación. Los tres algoritmos de resolución de laberinto han sido: [6]

- ***Búsqueda en amplitud:*** se trata de un algoritmo de búsqueda no informado que recorre un grafo comenzando por la raíz y eligiendo el primer nodo que tiene como hijo. A partir de ahí se van recorriendo el resto de nodos hijo hasta que ese nivel del grafo se ha completado. Una vez hecho eso se pasa al siguiente nivel empezando por el primer nodo hijo analizado y se repetirá el proceso de analizar todos los vecinos posibles. [8]

  Comienza escogiendo la primera celda que será la salida del usuario y comprueban las diferentes direcciones que puede tomar, como si estas fueran nodos hijos de la celda de salida, la cual hace de nodo raíz. Se comprueba si alguno de los hijos es la meta, ya que si es encontrada se detiene la búsqueda y la ejecución del algoritmo. Tras ello se repite el mecanismo hijo por hijo en el orden en el que han sido comprobados. [8]

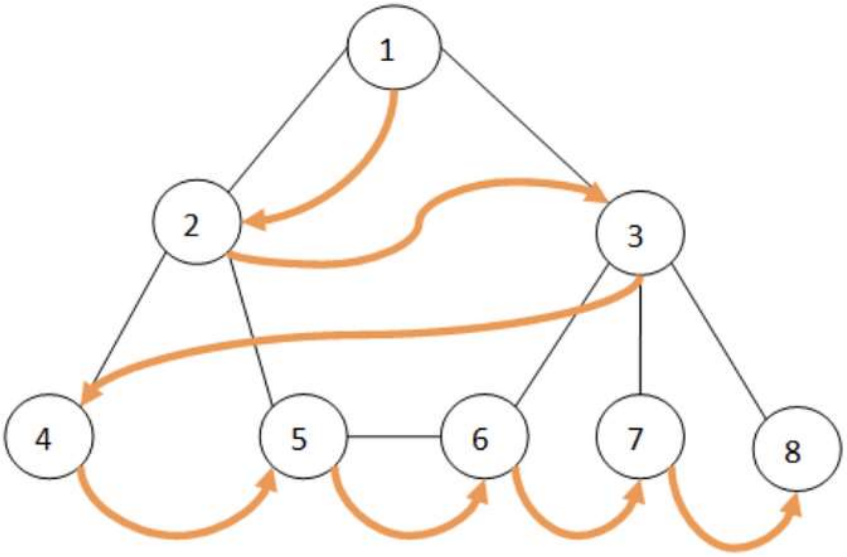

Ilustración 14 - Ejemplo visual de búsqueda en amplitud (8)

- ***Búsqueda en profundidad:*** es considerado el segundo algoritmo de búsqueda no informado utilizado para poder encontrar el camino entre dos nodos dentro de un grafo, realizando un recorrido en profundidad. Partirá de un nodo raíz e irá explorando el grafo hasta llegar al máximo nivel de profundidad. Cuando alcanza ese punto, el algoritmo recorre los vecinos de ese nivel hasta haber terminado, volviendo luego al nivel superior. Todos estos pasos irán repitiéndose hasta volver de nuevo al nodo raíz o haber encontrado la meta. [8]

  Se empieza con la celda que sirve como salida del usuario y se comprueba la primera dirección que puede tomar. Tras ello se coloca en esa nueva celda y realiza los mismos pasos que antes, de forma sucesiva hasta llegar a un caso en que no haya más direcciones posibles. Una vez hecho eso se recorren las celdas que tiene como otras direcciones en el último nivel y tras ello se vuelve al nivel superior. Se repite este proceso hasta alcanzar la meta, ya que una vez encontrada se detiene la ejecución del algoritmo.

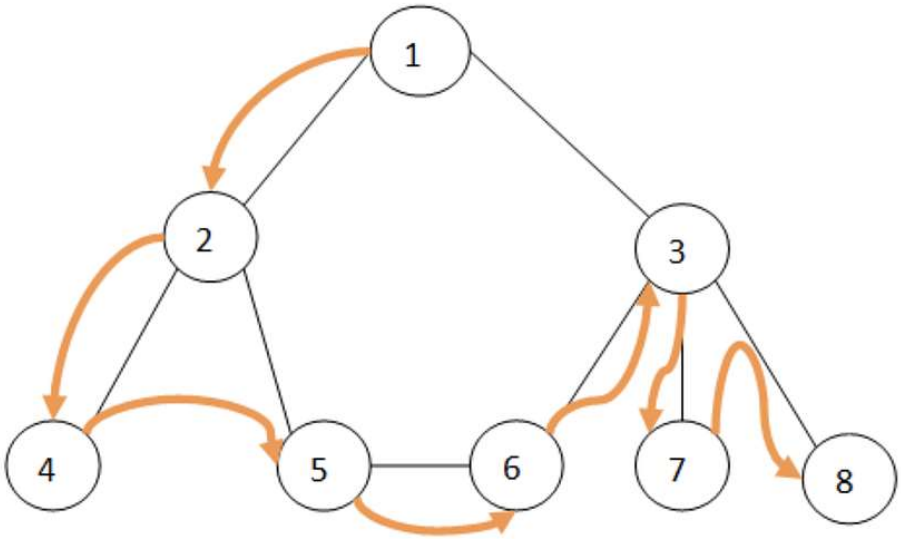

Ilustración 15 - Ejemplo visual de búsqueda en profundidad (8)

- ***Búsqueda A*:*** es considerado una mejora del algoritmo de Dijkstra para la resolución de laberintos, siendo el máximo exponente de la búsqueda con información. Consiste en construir la ruta desde un punto inicial a un nodo final usando la fórmula de las búsquedas heurísticas. Dicha fórmula dota a cada nodo de un valor y estará establecida por el programador. El algoritmo va a ir moviéndose a través de todas las celdas buscando el menor coste posible. [9]

  Tratándose de un método de búsqueda informada, si existe una solución siempre va a poder encontrarla, ya que va a ser considerado un algoritmo completo. No obstante, el tiempo que pueda tardar en encontrar solución va a depender de la fórmula que se escoja para darle valor a cada uno de los nodos. [9] [10]

$$F(n) = g(n) + h(n)$$

*Ecuación 1 - Ecuación general para A**

La ecuación 1 representa la ecuación básica heurística para este tipo de algoritmo. La F es el coste total que recibe ese nodo que se está explorando, que se obtiene con la suma de dos valores: g y h. El elemento g es el valor del camino de un nodo a otro por poder moverse y el elemento h el valor de la heurística escogida para el algoritmo. En todo momento n va a hacer referencia al nodo o celda que se está tratando. [10]

Gracias a estos tres algoritmos diferentes de búsqueda se puede llegar a ver cómo varían unos métodos de otros. Los tres se pueden traducir en la búsqueda a través de un grafo, por lo que se deberá de concebir el laberinto como un grafo en sí mismo.

4.6- IMPLEMENTACIÓN DE ALGORITMOS

El primer paso, después de hacer el laberinto, es almacenarlo en forma de grafo. La implementación de cada algoritmo necesita interpretar el laberinto como si fuera un grafo y crear un camino desde el inicio a la meta. Cada algoritmo encontrará este camino de una forma, siendo este punto lo que diferencia un método u otro. No obstante, la implementación general de toda la aplicación quedará dividida en 3 pasos:

- ***Transcripción a grafo:*** lectura del laberinto como si fuera un grafo, teniendo a las celdas por nodos y los caminos entre ellas como las relaciones de vecindad o de padre/hijo. Se escoge una casilla como inicio y se explora sus hijos, elaborando las conexiones entre ellos, siempre en el sentido de las agujas del reloj. Tras ellos se repite el mismo procedimiento con cada uno de ellos hasta que no haya más relaciones.

  Esta funcionalidad está implementada en un método llamado crearGrafo, el cual contiene una cola que se inicia con el nodo que sirve como primera celda y una lista de diferentes celdas con la información de sus relaciones. Empezando en la celda [0,0] se irán añadiendo a esta estructura todos los hijos del primer nodo que haya y una vez acabado, se pasa al siguiente. Este proceso se repite hasta incorporar todas las celdas del laberinto a la estructura.

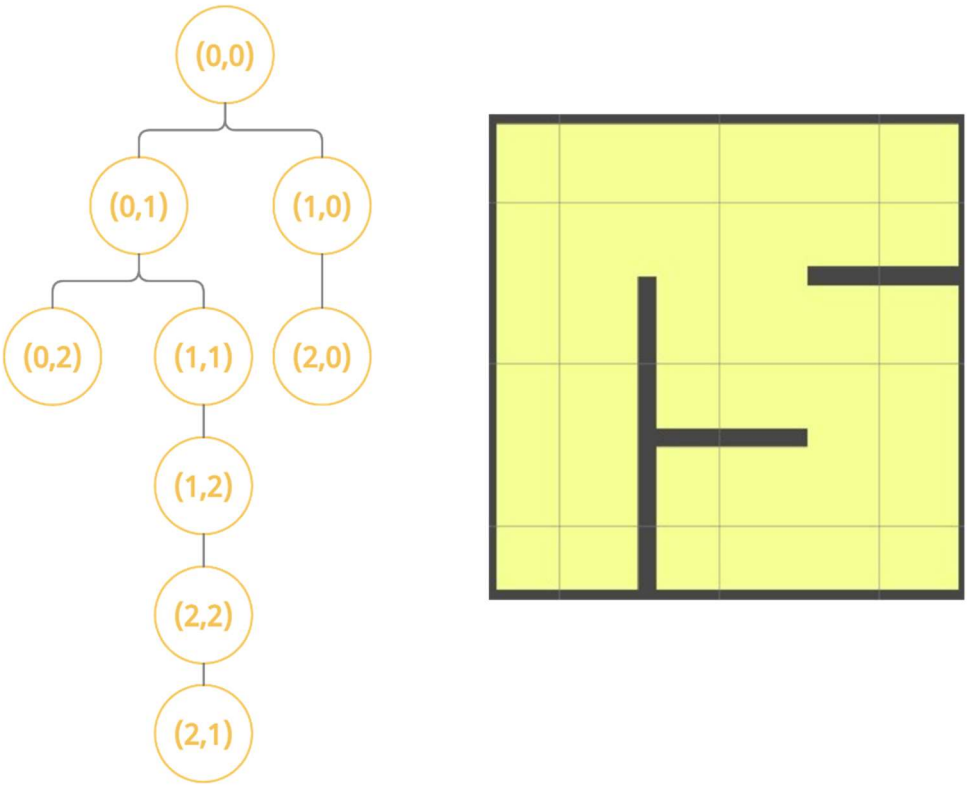

Ilustración 16 - Ejemplo de transformación a grafo

- ***Exploración del grafo:*** se han implementado los siguientes algoritmos para la búsqueda de la salida del laberinto:
    - **Recorrido en amplitud:** en la ilustración 17 se representa el recorrido en amplitud del laberinto mostrado en la ilustración 16.

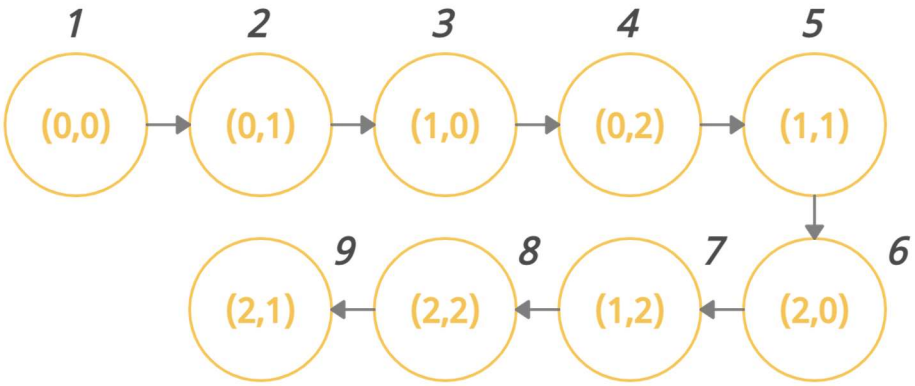

Ilustración 17 - Ejemplo orden de nodos en búsqueda por amplitud

Bajo el nombre de busquedaEnAmplitud, existe un método dentro del programa que realizará la recogida de los nodos del laberinto con este tipo de algoritmo. Este método va a ser realizado gracias a una estructura de

datos del tipo cola que contendrá las diferentes celdas del laberinto con sus conexiones. Los pasos que irá tomando serán los siguientes:

1- Se mete en la estructura de datos la celda que sirve como inicio.
2- Se inicia una variable como contador a 0.
3- Se comienza un bucle *while* el cual no llegará a su final hasta que la variable que sirve como contador alcance el número total de celdas, es decir, recorra todo el laberinto.
4- Dentro del bucle se explorarán todos los hijos de la primera celda que esté en la cola y se añadirán a la misma en ese orden.
5- Una vez termine de introducir todos los hijos saca la primera celda, que será la que se acaba de explorar y se suma una unidad al contador previamente creado.

Para finalizar el método se devuelve la cola con el orden correcto de recorrido del grafo al código principal. Se consigue una estructura de datos que representa el orden establecido en la ilustración 17.

- **Recorrido en profundidad:** a través de la ilustración 18 se representa el recorrido en profundidad del laberinto mostrado en la ilustración 16.

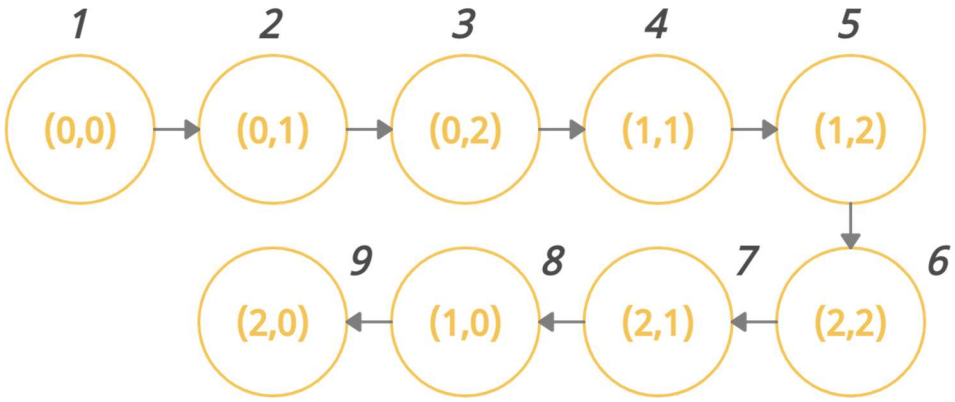

Ilustración 18 - Ejemplo orden de nodos en búsqueda en profundidad

Para este tipo de algoritmo se ha decidido recurrir a un método que va a funcionar con recursividad. El método recibe el nombre de busquedaEnProfundidad y necesitará contar con una estructura de datos de tipo cola que contiene las celdas del laberinto, además de la celda que se está analizando en cada momento. Los pasos que va a seguir el método serán:

1- **Caso base:** si la celda que se está explorando actualmente no tiene hijos y no se trata de la celda raíz, se introduce en la cola de celdas y se devuelve toda la estructura de datos.
2- **Resto de casos:** se comienza introduciendo en la estructura de datos la celda actual que se está explorando y tras ello se inicia un bucle *for* de todos sus hijos. Empezando de izquierda a derecha, se realizará la llamada a la recursividad por cada uno de los hijos. Cuando todo el bucle termina y se vuelve de la recursividad, se devuelve la estructura de datos, al igual que en el caso base.

El método devolverá una estructura de datos del tipo cola llena de celdas ordenadas como si se hubiera recorrido el grafo con el algoritmo de búsqueda en profundidad, como representa la ilustración 18.

- **Recorrido en A estrella:** va a tratarse de la implementación más costosa de las tres, ya que ordenará los grafos a través de una variable nueva definida por el programador. Se debe conocer la meta para realizar los cálculos que facilitarán el algoritmo y el cálculo de esta nueva variable. Va a quedar definida a través de dos costes, los cuales reciben el nombre de coste H y coste G.

  Para el primero se ha decido utilizar la distancia Manhattan. El segundo coste se centra en mantener el número de pasos que llevan hacia esa celda desde el inicio. En caso de empate de coste, el primer nodo que haya entrado en la estructura que va guardando y ordenando los nodos será el que prevalezca. Con las ilustraciones 19 y 20 se explican los costes que se encuentran en el laberinto de la ilustración 16 y el orden en el que se recorrerán.

Celda	Coste H	Coste G	Coste A
(0,0)	4	0	4
(0,1)	3	1	4
(0,2)	2	2	4
(1,0)	3	1	4
(1,1)	2	2	4
(1,2)	1	3	4
(2,0)	2	2	4
(2,1)	1	4	5
(2,2)	0	4	4

Ilustración 19 - Ejemplos de gastos de cada celda

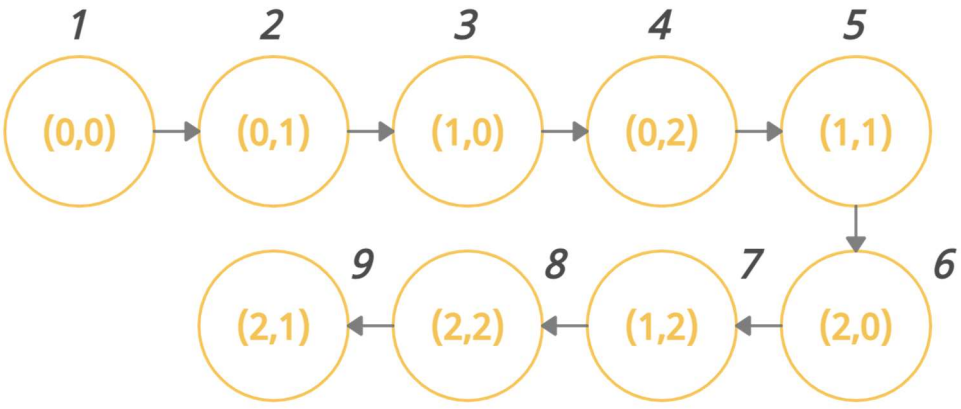

Ilustración 20 - Ejemplo orden de nodos en búsqueda A Estrella

El algoritmo A Estrella está implementado en un método llamado busquedaEnAEstrella. Se mantiene el uso de un método que usa recursividad, el cual sigue recogiendo en su cabecera tanto una estructura de datos de tipo cola con las celdas del laberinto y la celda actual que se está explorando en cada momento. Los pasos que va a seguir serán los mismos que en la búsqueda en profundidad, pero dentro del bucle *for*

donde se exploran los hijos, estos estarán previamente ordenados por su coste total. Con eso se consigue una cola de celdas con el orden de la ilustración 20.

Todos estos métodos tendrán en común que van a recoger, en una misma estructura de datos de forma ordenada, todas las celdas de un laberinto. De esta forma, cuando se analiza cada una de ellas de forma visual es mucho más sencillo de mostrar y queda totalmente aislado de toda la programación y elaboración del algoritmo.

- ***Comprobación de resultado:*** el último paso a seguir es dejar que el usuario sea capaz de comprobar qué hace un algoritmo para encontrar la salida. Todo va a quedar reflejado en una interfaz intuitiva para los usuarios, como si se tratase de un panel de navegación con paradas cuando se necesite.

  En el lado izquierdo de la pantalla se observa un panel de navegación con botones e información de la situación en la que se encuentra el usuario, mientras que en el lado derecho se encuentra la información del algoritmo y el laberinto, marcando de forma visual la localización actual:

  - **Panel de navegación:** se encuentra en la parte izquierda de la pantalla, con dos botones y diferentes textos que aportan información escrita de la situación del usuario. Los dos botones se centran en el avance o el retroceso de la realización del algoritmo, programados de tal forma que, si no hay opción de retroceder, este botón desaparezca. Si se ha llegado a la meta desaparecen los dos botones y termina la navegación.

    La parte de texto tiene dos partes fundamentales: información general e información de las celdas. Por un lado, la información general se trata de la meta que se está tratando de alcanzar en todo momento y la información del paso en el que está el usuario. Por otro lado, se encuentra la información de las celdas, un apartado que va indicando la celda en la que se encuentra el usuario, la celda de la que viene y la próxima celda que va a analizar o que está analizando.

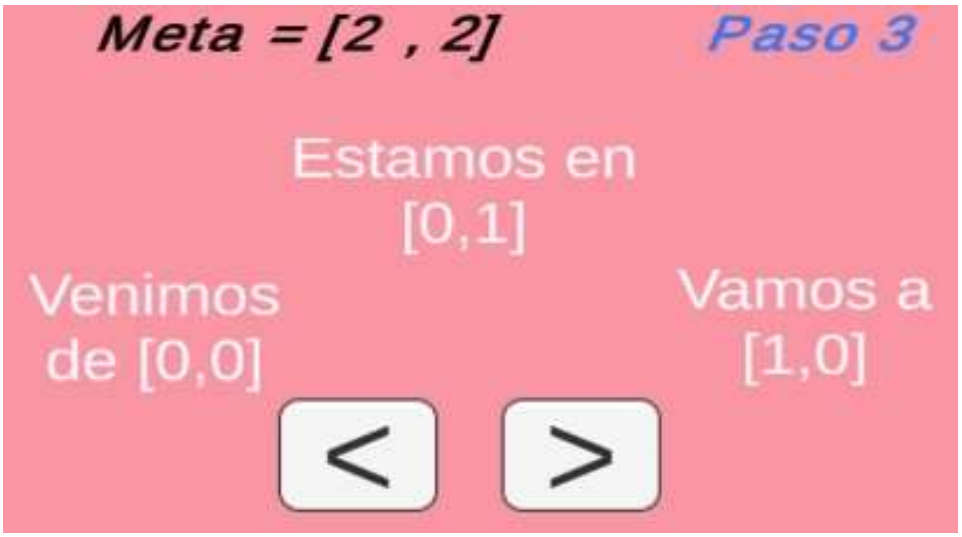

Ilustración 21 - Panel de navegación

- **Panel de visualización:** se muestra en código de Python la forma en la que está funcionando el algoritmo de búsqueda escogido. A través de imágenes y texto se irán estableciendo tres situaciones diferentes para cada uno de los algoritmos, como si de una máquina de estados se tratara.

  El primer estado es el base, es decir, que el algoritmo se encuentre en la meta o lo que se ha marcado como meta. Para todos los casos se establece la esquina contraria al [0,0], pero se podría llegar a modificar para que pudiera ser otra celda. En el caso de que se cumpla se para toda la búsqueda. Durante este momento la celda se marcará de color amarillo.

  Al no cumplirse el caso base se pasa al resto de casos. Será en este apartado donde cada algoritmo presentará sus diferencias para ver qué celda recorrer en el próximo caso. Se mantiene el color amarillo del primer estado en la celda escogida.

  Para el último estado que se analiza se realiza el concepto de recursividad, ya que se ha analizado la celda actual y se pasa a la siguiente. Se volverá al primer estado y la celda que se ha estado analizando hasta ese momento pasará a entrar en el color verde, ya que quedará como visitada.

Ilustración 22 - Ejemplo de panel de visualización

Durante todo momento, el usuario podrá ir navegando hacia delante o hacia atrás en todos los estados que se han analizado. Cuando se alcanza la meta, todo lo que tiene ver con la interfaz desaparece y se muestra una pantalla resumen con la opción de analizar el mismo laberinto en los otros algoritmos.

Ilustración 23 - Resumen tras comprobación

4.7- SELECCIÓN DE DISEÑO VISUAL

Al tratarse de una aplicación donde prima las interfaces por encima de otro tipo de elemento visual, se ha tratado de escoger un estilo sencillo y que presente una correcta cohesión. Se ha decidido usar tilemaps, sprites e imágenes que están libres de derechos y siendo enteramente modificadas. Todos los elementos visuales que se han decidido utilizar en la estética han sido elaborados usando otros recursos adquiridos a través de diferentes librerías gratuitas, pero enteramente modificados en Adobe Photoshop para poder adaptarlo.

Los elementos de la decoración gráfica están inspirados en una estética de carácter industrial. Se representan elementos gráficos que están relacionados con el mundo de los

robots y las máquinas. Los tonos grises y amarillos van a predominar en toda la paleta de colores elegida.

4.8- ELABORACIÓN DE COMPONENTES VISUALES

Aunque la elaboración de cada elemento visual ha sido realizada de una forma diferente, se ha elegido una fuente común con licencia gratuita por Typodermic Fonts llamada Karma Suture [11]. Observando cada interfaz por separado, se han realizado los siguientes elementos visuales:

- ***Pantalla de Título***: solo presenta tres elementos de diseño que han sido elaborados: el fondo de la aplicación, el cartel que indica el título de la aplicación y el botón que da comienzo.

  El fondo se ha realizado de forma simple y útil para el resto de pantallas que aparecen en la aplicación. La imagen del título crea los precedentes para los próximos carteles, escogiendo la paleta de colores que se van a repetir. Por último, el botón de comenzar presentará el color que predominará en la aplicación: el amarillo. Con un simple vistazo se pude ver el tono que tendrá toda la aplicación.

Ilustración 24 - Imagen final de la pantalla de título

- ***Pantalla de Selección:*** en esta segunda interfaz habrá más elementos interactivos y visuales que deben de implementarse. Lo primero es el título de la aplicación, a modo informativo. Junto a ello hay dos menús desplegables para que puedan llegar a elegir el algoritmo por el que quieren comenzar y las dimensiones. Lo último de esta interfaz es el botón que daría comienzo a la visualización de todo el algoritmo.

Buscando una mayor implicación en la elaboración del diseño, se ha decidido obviar cualquier tipo de *placeholder* que presenta Unity y crear todos los elementos, incluyendo símbolos de los menús desplegables y su estética dentro y fuera. De esta forma se han ido creando cada uno de los símbolos, cuadrados y marcos.

Ilustración 25 - Imagen final de la pantalla de selección

Ilustración 26 - Imagen final de desplegable personalizado

- ***Pantalla de Visualización:*** tanto en la parte de navegación, con sus respectivos datos de estado, como en la parte del laberinto, habrá que elaborar una serie de elementos visuales para que el usuario pueda entender a la perfección lo que está viendo por pantalla.

  Los botones de la parte inferior de la pantalla presentan las funciones básicas de esta interfaz. Serán tres botones: uno para crear el laberinto, otro para resolver el

laberinto y otro para visualizar la solución. Todos ellos han sido elaborados bajo el mismo patrón, tratando de diferenciarse entre sí a través de la imagen que los caracteriza y el texto que tratan de mostrar.

Ilustración 27 - Imagen final de la pantalla de visualización

Una vez elaborado el laberinto, aparece por pantalla y se ven dos elementos claves dentro: la salida y la meta. El punto de partida será representado con una bandera verde, mientras que la meta será una bandera de cuadros blancos y negros. Para el laberinto se han elaborado 16 versiones de celdas de laberinto que aparecen según la codificación que tengan y su número de conexiones.

A la vez que el laberinto también aparece el panel de navegación en la parte izquierda de la pantalla. Esta parte permite avanzar y retroceder según el momento del algoritmo de búsqueda que se esté realizando para poder entender todo el código y las elecciones del mismo. Gracias a tres huecos que indican la celda actual, la anterior y la siguiente, se podrá saber dónde se encuentra el usuario, además de los botones de avance y retroceso o los elementos que indican el paso en el que se encuentran y la meta establecida.

Ilustración 28 - Imagen de panel de navegación final

Lo último será la parte donde el usuario podrá ver el código que se está analizando en cada paso del algoritmo. Para que todo se entienda mejor, se ha incluido una funcionalidad con la que se puede ver tanto el fragmento de código que se está analizando, como la explicación larga del mismo, con el fin de ayudar mucho más al usuario. En todo momento el usuario podrá acceder a esta explicación y entender lo que está ocurriendo con lo que podría ser comentarios dentro del código.

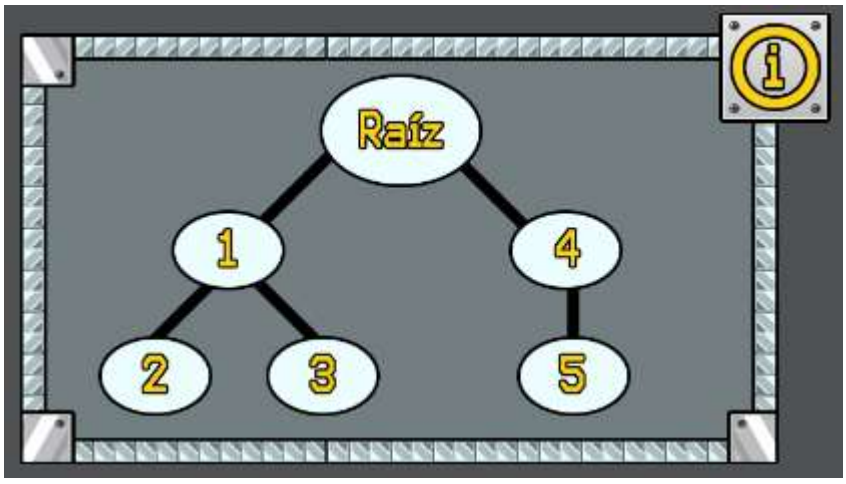

Ilustración 29 - Imagen final de zona código

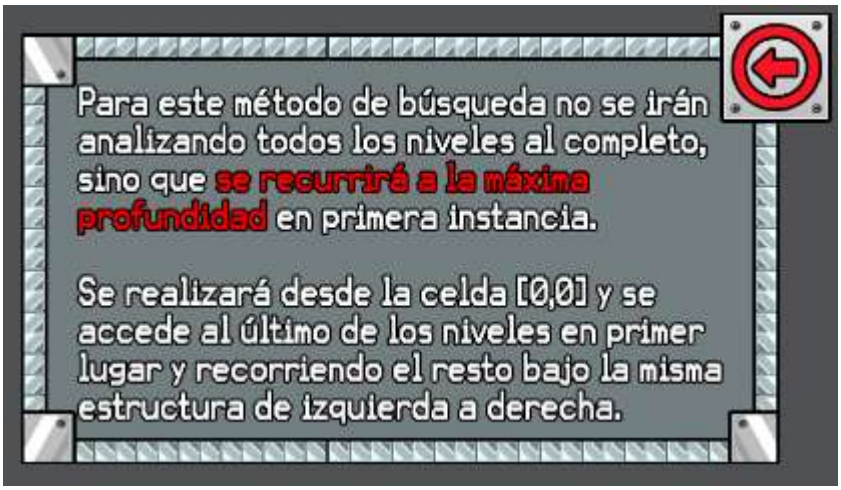

Ilustración 30 - Imagen final de zona de información explicada

Un detalle que ayuda mucho a dar mayor cohesión a todo lo de seguir un camino a través del laberinto es ir indicando las celdas que se van visitando. Para ello se ha implementado una imagen que va marcando el camino que se toma hasta llegar a la meta. Cuando se alcanza dicha meta desaparece toda la interfaz y aparece una imagen resumen que permite poder cambiar de algoritmo, para probar el mismo laberinto con otros métodos.

Ilustración 31 - Imagen final laberinto recorrido

Ilustración 32 - Imagen final resumen de visualización

Para mantener una gran conexión entre toda la aplicación se han repetido colores y marcos para los diferentes botones, de forma que la aplicación presenta una cómoda navegación. No obstante, se podría llegar a extrapolar a muchos estilos diferentes e, incluso, cambiar los colores y que sigan funcionando igual de bien.

4.9- ELEMENTOS ADICIONALES

El hecho de ir actualizando este tipo de trabajos estará a la orden del día e irán surgiendo muchos elementos que podrían aparecer según se avanza o el propio creador se percata de ello.

Dentro de esta aplicación, siguiendo las recomendaciones de los tutores, se han decidido implementar mejoras estéticas. Cada uno de estos cambios ha sido implementado gracias a la combinación de nuevos diseños visuales que van a verse por pantalla y pequeños cambios en el código, que no han trastocado la estructura general del programa. Para poder analizarlos correctamente los desglosamos de forma individual:

- ***Cambio en las dimensiones del laberinto:*** en una primera instancia se decidió escoger tres tipos de dimensiones para el posible laberinto que se iba a crear por pantalla. Siendo la primera de ellas la de 3x3, que ha sido la más utilizada para ejemplificar el funcionamiento de la aplicación, las otras dos dimensiones que se escogieron fueron múltiplos directos de esta, pasando a 9x9 y 18x18. Aunque se consiguió un resultado que funcionaba a la perfección, las grandes dimensiones de estos laberintos no permitían realizar cómodamente las comprobaciones y hacían estos trabajos bastante largos.

Debido a todo esto, se decidió realizar cambios en las dimensiones que se podrían escoger para los laberintos, ya que el único cambio que debía realizarse se implementaría en el código. Ante esto, las nuevas dimensiones elaboradas serán 4x4 y 6x6, útiles para poder ver algunas situaciones importantes y con una dimensión acorde con el espacio de interfaz que va destinado a ello.

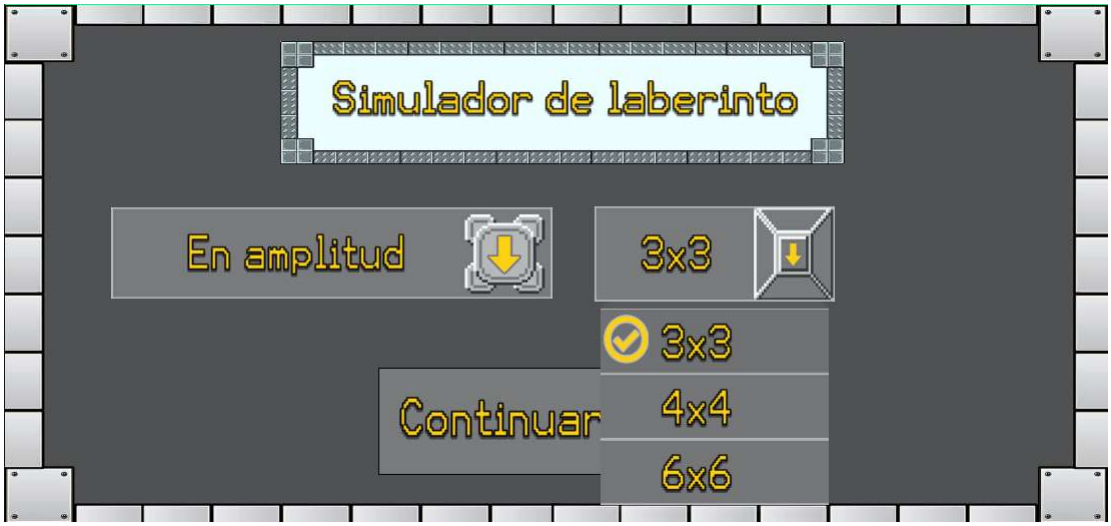

Ilustración 33 - Modificación de dimensiones del laberinto

- ***Resumen final mostrado en el laberinto:*** se ha planteado una ventana final a modo de resumen donde se indica cuántos pasos ha hecho el algoritmo que se ha escogido para encontrar la meta. Se ha buscado mostrarlo de una forma más visual para poder entenderlo, donde se pueda replicar los caminos innecesarios que ha tomado.

  Se ha partido de las imágenes que simbolizan los pasos realizados dentro del laberinto y con un código de colores verde/rojo para indicar el camino hacia la meta y los intentos innecesarios se muestra por pantalla.

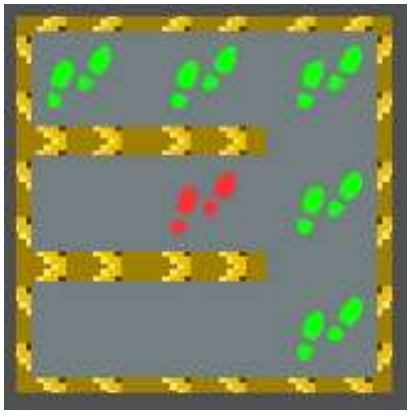

Ilustración 34 - Ejemplo de resumen de pasos en laberinto

- ***Información de las opciones de A Estrella:*** con el fin de poder entender mucho más el tercer algoritmo tratado en esta aplicación, se ha decidido poder explicar mejor el proceso de elección para A Estrella. Cada vez que se hace vigente la llegada a una celda nueva, se despliegan los posibles hijos que tiene con los costes de avanzar a sus casillas, de forma que se puede ir adelantando cuál será el más prometedor.

  Se ha decidido utilizar las dos variables, tanto h como g, pero cuando hay empate en la suma de estos valores, el algoritmo escogerá el primero que haya llegado a la estructura de celdas que recogen todas.

Ilustración 35 - Ejemplo de gastos de celdas en A Estrella

Estas decisiones mejoran mucho la experiencia del usuario dentro de la aplicación y permiten un mejor entendimiento de lo que se está mostrando por pantalla. Algunos de los cambios realizados, de forma adicional, han ido destinados a depurar todo el código y no dejar ningún fallo, los cuales solo se centran en la programación del trabajo.

5

RESULTADOS

Para analizar cómo se ha tratado el funcionamiento final de la aplicación y ver resultados técnicos, los resultados deseados han sido divididos en dos enfoques:

- ***Depuración de bugs:*** se han realizado ensayos de diferentes momentos por los que puede pasar la aplicación y valorar los resultados que ha tenido. Ante esta situación, comprobar visualmente puede no ser suficiente, llegando a usar dos métodos para depurar el código y comprender mejor la resolución de problemas:
    - **Puntos de ruptura:** han sido utilizados con el fin de conocer el flujo de la aplicación, tratando de comprender que los algoritmos de búsqueda eran ejecutados en el orden que debían realizarse, ya que al organizar todo en estructuras de datos abstractas, resultaba mucho más útil.
    - **Mensajes por consola:** se implementaron para poder tener una forma más visual del momento exacto en el que se encuentra un programa según las decisiones de usuario, ya que se muestra a través de un mensaje que el programador puede ver.

  Desde el punto de vista de programación han resultado herramientas muy precisas para depurar pequeños *bugs* que podrían encontrarse al tratar de poner la aplicación en límites fuera de la normalidad. No obstante, el entreno y uso de algún tipo de comunidad o personas externas, podría generar un *feedback* mucho más completo.

- ***Análisis de rendimiento de la aplicación:*** se ha buscado analizar el rendimiento real dentro de un ordenador con el uso de esta aplicación, ya que se va a tratar de compartir con fines educativos. Para poder realizar esto, el propio motor de Unity presenta una herramienta muy completa que desgrana el uso de los recursos del ordenador donde se ejecuta la aplicación, analizando el uso dedicado de CPU.

Esta herramienta es Unity Profiler que analiza diferentes elementos técnicos y que en esta aplicación ha podido recoger lo siguiente:

- **Uso de CPU reducido a lo esencial:** el uso de la CPU queda reducido, no llegando a alcanzar los 30 FPS de forma rutinaria, salvo en momentos donde se juntan muchos elementos por pantalla. Comprobando los campos en los que destina la CPU, todos sus esfuerzos van a cuatro áreas diferentes: rendering, UI, script y otros. La sección de otros engloba todo lo que puede tener relevancia con el editor de Unity, algo que realmente podemos dejar de lado ya que en la ejecución final no se verá.

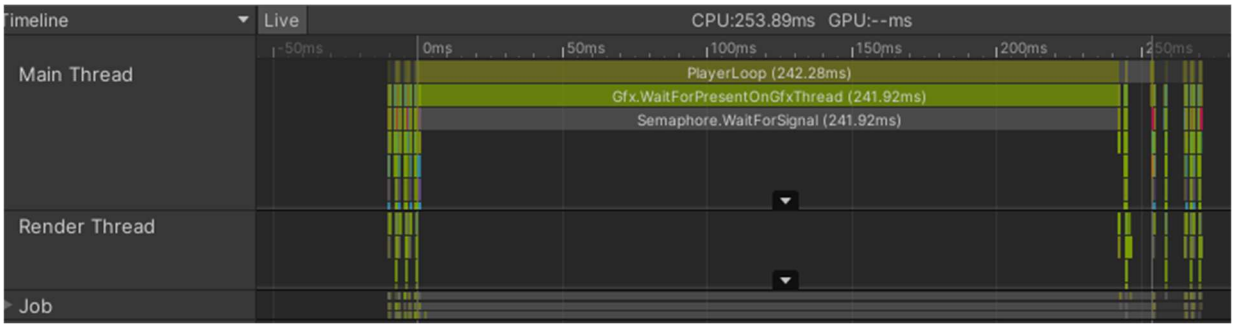

Ilustración 36 - Ejemplo momento de mayor uso de CPU

- **Poca memoria utilizada para su ejecución:** al momento de realizar este análisis, se destinará más memoria a la parte de gestión y control de la aplicación, siendo enteramente seguida por los gráficos, ya que presenta mucho contenido visual.

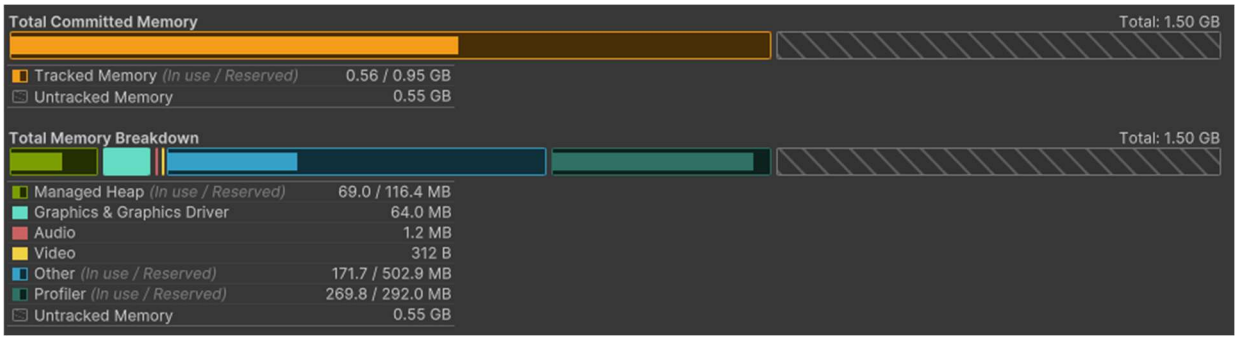

Ilustración 37 - Ejemplo memoria utilizada

- **Esfuerzos destinados únicamente a las interfaces:** esto hace que no haya nada de interacción o cómputo por parte de la GPU, sino que en todo momento se encarga la CPU de los cambios oportunos en lo que se debe de ver por pantalla.

No utilizar físicas ni elementos que someten a más esfuerzo a Unity, hace que la aplicación sea manejable y no precisa de altas prestaciones en un ordenador para ejecutarla.

6

CONCLUSIONES

Revisando todos los objetivos que se han marcado desde un primer momento para este trabajo, se puede conseguir un elevado grado de satisfacción, ya que se puede decir que se han conseguido ejecutar correctamente. No se ha conseguido realizar de la forma más completa o perfecta posible, ya que, al tratarse de un trabajo realizado por un único integrante, la carga es mucho mayor, pero desglosando en dos partes todos los objetivos planteados, se puede llegar al siguiente resumen:

- ***Parte técnica:*** englobando dentro de este apartado todo lo que tiene que ver con la programación, se afirma que se han conseguido llevar a cabo correctamente todos los objetivos específicos que conforma el objetivo principal de "Programación de diferentes algoritmos de búsqueda para resolver la salida de laberintos".

  Con el resultado final se puede ver que la parte centrada en la programación y el control del entorno de desarrollo ha sido mucho más eficiente que lo que tiene que ver con la parte artística. No obstante, el objetivo de la aplicación consiste en dar a entender a futuros alumnos cómo funcionan los algoritmos, algo que debe ser correctamente programado para que no genere ningún gran fallo.

  Habiéndose planteado solo tres algoritmos se han implementado correctamente, aunque con unos esquemas propios del programador para ayudar a la visualización posterior. Llegando a darse otro tipo de algoritmos este tipo de estructura deberá de plantearse de otra manera, aunque para el trabajo funciona correctamente.

- ***Parte artística:*** sin tratarse de un apartado de la aplicación en lo que se busque destacar, se ha conseguido obtener un resultado adecuado para que la aplicación destaque. Hay muchos elementos que podrían mejorar y gestionarse de otra forma,

sobre todo en cuanto a distribución, colores o diseños, pero se entiende lo que se quiere transmitir.

A lo largo de toda la realización de la aplicación se ha decidido dar más importancia a la parte de programación por encima de la artística, algo que ha quedado más que plasmado, siendo claramente un punto a mejorar en caso de contar con un equipo mayor para perfeccionar todo este trabajo.

7

BIBLIOGRAFÍA

(1) Cabot J. Visualización Animada de las Estructuras de Datos más conocidas. Ingeniería de Software. Programación. Luxemburgo. 2018.

(2) Maida, E.G., Pacienzia, J. Metodologías de desarrollo software. Tesis de Licenciatura en Sistemas y Computación. Facultad de Química e Ingeniería. Universidad Católica Argentina, 2015.

(3) Miguel Diaz, F. Conociendo Unity 5. Desarrollo de Software. Aplicaciones Web. DEV. Paradigma. Madrid. 2018.

(4) AcademiaAndroid. Canvas: creación e interfaz de usuario en Unity. Nivel Básico, Videojuegos. Tutoriales. 2015.

(5) TRINIT. Tutorial: Diseñar un interfaz de UI en Unity 2019. Clases Online. Master Videojuegos Online. Asociación de Informáticos de Zaragoza. Zaragoza. 2019.

(6) Hybesis. Pathfinding algorithms: the four Pillars. H.Urna. Medium. 2020.

(7) Tomás Mariano, V.T., Pozas Cárdenas, M., Hernández Camacho, J. Propuesta para la generación de laberintos ampliados en 2D. Ciencia Huasteca Boletín Científico de la Escuela Superior de Huejutla. Vol.1 Núm.1. Universidad Autónoma del Estado de Hidalgo. 2013.

(8) López Mamani, M. DFS vs BFS. Blog Encora. Ingeniería y Desarrollo de Productos. Encora Digital Inc. California. 2020.

(9) Giménez, G. Métodos de Búsquedas en Inteligencia Artificial. Distintas Herramientas Tecnológicas en la Actualidad. Universidad Nacional Experimental Politécnica de la Fuerza Armada Nacional. Ministerio del Poder Popular para la Nación. Venezuela. 2018.

(10) Aymar Marza S. et al. Inteligencia Artificial (Búsqueda A Estrella). Facultad de Ciencais y Tecnología. Universidad Mayor de San Simón. Bolivia. 2019.

(11) Karma Suture Font. Typodermic Fonts. 1001 Free Fonts. Disponible online en: https://www.1001freefonts.com/karma-suture.font