ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS

Curso Académico 2019/2020

Trabajo Fin de Grado

Desarrollo de una herramienta para la visualización de algoritmos de ordenación

Autor: Juan Pedro Guirado Sánchez

Directores: Juan José Pantrigo Fernández

Agradecimientos

En primer lugar, a mi tutor Juan José Pantrigo Fernández por su interés desde el primer instante y por su ayuda constante y cordial.

En segundo lugar, a mi familia y mi pareja, los cuales me han animado en todo momento y motivado en los momentos más difíciles.

Por último, a mis amigos, los cuales también me han ayudado a darle forma a este proyecto.

Índice

Tabla de figuras

Resumen

V-Algo es un proyecto de desarrollo que consiste en la creación de una aplicación para la visualización de algoritmos de ordenación. Surge de la idea de apoyar de forma gráfica los contenidos que se dan en la asignatura de Algoritmos para Juegos y del interés por mi parte en este campo.

Es una aplicación desarrollada en Unity y cuyo código está escrito en C#. Su plataforma principal de lanzamiento es la Web gracias a la sencilla exportación que este motor proporciona mediante la API WebGL.

La aplicación se encuentra alojada en una Git Hub Page [7] totalmente publica, por lo que cualquiera puede acceder a ella y ejecutarla sin ningún requerimiento especial, más allá de que nuestro navegador sea compatible con WebGL. Este aspecto es crucial para cumplir uno de los objetivos principales del proyecto, que sea accesible para cualquier usuario que quiera estudiar los algoritmos presentes y que la herramienta le apoye de forma gráfica en este cometido.

Los algoritmos presentes en la herramienta son diferentes soluciones al problema de la ordenación y son los siguientes: burbuja, inserción, selección, *merge sort* y *quick sort*. Antes de la ejecución de cada algoritmo, es posible elegir el número de elementos que tendrá el array a ordenar y este se creará de forma aleatoria en cada ejecución.

La ejecución dispone de 2 modos: automático y paso por paso. En este último se puede avanzar un paso adelante o atrás en la ejecución. Es posible cambiar entre estos modos pulsando el botón de *play*/pausa. En cualquier momento el usuario puede moverse al principio o al final de la ejecución, además de poder cambiar la velocidad de esta.

Por último, es posible elegir entre 2 idiomas dentro de la aplicación: español o inglés.

Palabras clave

Algoritmo, Ordenación, Burbuja, *Merge Sort*, *Quick Sort*, *Insertion Sort*, *Selection Sort* Unity, *WebGL*, V-Algo

Capítulo 1 – Introducción

V- Algo surge de la idea de apoyar de una forma gráfica la asignatura de Algoritmos para juegos. Explorando ciertos videos en Internet que representaban de una forma visual como funcionaban distintos algoritmos de ordenación, surgió la idea de que una aplicación interactiva desarrollada en Unity podría ser interesante como TFG.

Es necesario remarcar la importancia de las herramientas de visualización de algoritmos, sobre todo en la docencia, para ayudar a los alumnos a comprender cómo funcionan realmente los algoritmos presentes en todo tipo de software en la actualidad. Supone un buen complemento para apoyar de forma visual el código y mejorar las herramientas de depuración y visualización que nos ofrece el propio lenguaje o el entorno de desarrollo.

1.1 Algoritmos considerados

Los algoritmos que hemos considerado para su inclusión en el proyecto son 5: ordenación por burbuja o *bubble sort*, ordenación por mezcla o *merge sort*, ordenación rápida o *quick sort*, ordenación por selección o *selection sort* y, por último, ordenación por inserción o *insertion sort*.

En primer lugar, se ha decidido incluir el algoritmo de la burbuja por ser uno de los más sencillos a la vez que menos eficientes y más famosos, siendo una alternativa interesante a considerar. En segundo lugar, se han incluido los algoritmos de ordenación por mezcla y ordenación rápida, por tratarse de alternativas recursivas y presentes en el temario de la asignatura. Por último, para enriquecer el trabajo con alguna alternativa más, se han incluido los algoritmos de ordenación por selección e inserción, alternativas con planteamiento similar, aunque con diferencias notables.

Se ha decidido que todos los algoritmos sean de ordenación para tratar una temática única y poder ver más alternativas a un mismo problema. Para trabajos futuros queda la inclusión de otro tipo de problemas y los algoritmos destinados a solucionarlos, de lo cual hablaremos con más detalle en la sección 6.2.

Capítulo 2 - Estado del arte: alternativas presentes

2.1 Visualgo

Visualgo [1] es un proyecto de la universidad de Singapur, el cual comenzó en 2011. El concepto fue propuesto por el Dr Steven Halim como una herramienta para ayudar a sus alumnos a comprender el funcionamiento de los algoritmos y estructuras de datos. Esta aplicación es una alternativa muy completa en cuanto a número de algoritmos desarrollados para visualizar, como se puede ver en la Figura 1. Sigue en desarrollo y según la aplicación, algunos de los algoritmos presentes sólo se encuentran para ser visualizados en VisuAlgo.

Figura 1: Algoritmos presentes visualgo. Fuente: Visualgo [1]

Su mayor defecto es el apartado visual, con una usabilidad deficiente. Se producen ciertas combinaciones de colores que no permiten ver los textos con claridad. Este problema se acusa más en la visualización del código en ejecución, complicando la comprensión de este, como se puede apreciar en la Figura 2.

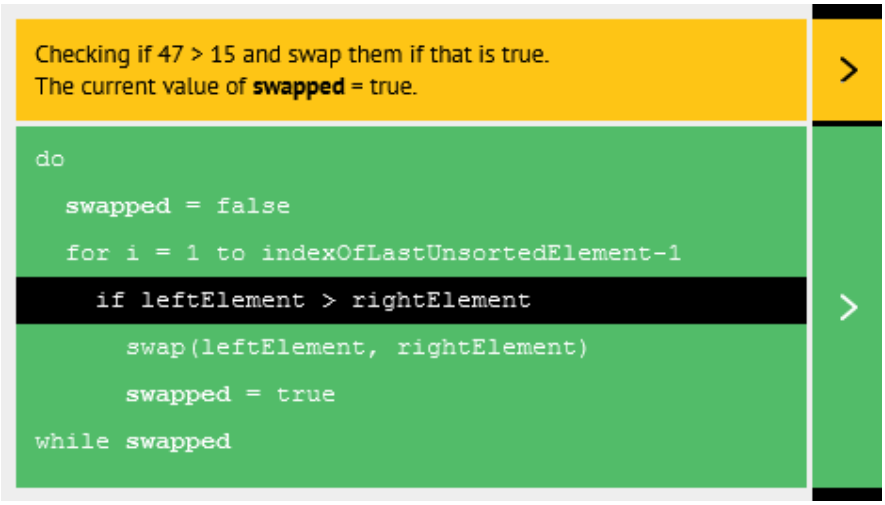

Figura 2: Visualización de código en visualgo. Fuente: Visualgo [1]

Consideramos que el aprovechamiento en pantalla no es el más adecuado y que la navegación por los apartados podría mejorarse. Se presenta demasiada información simultáneamente al usuario, en grandes bloques de texto, lo que pensamos que podría provocar la perdida de interés. Esto se puede apreciar en la Figura 3

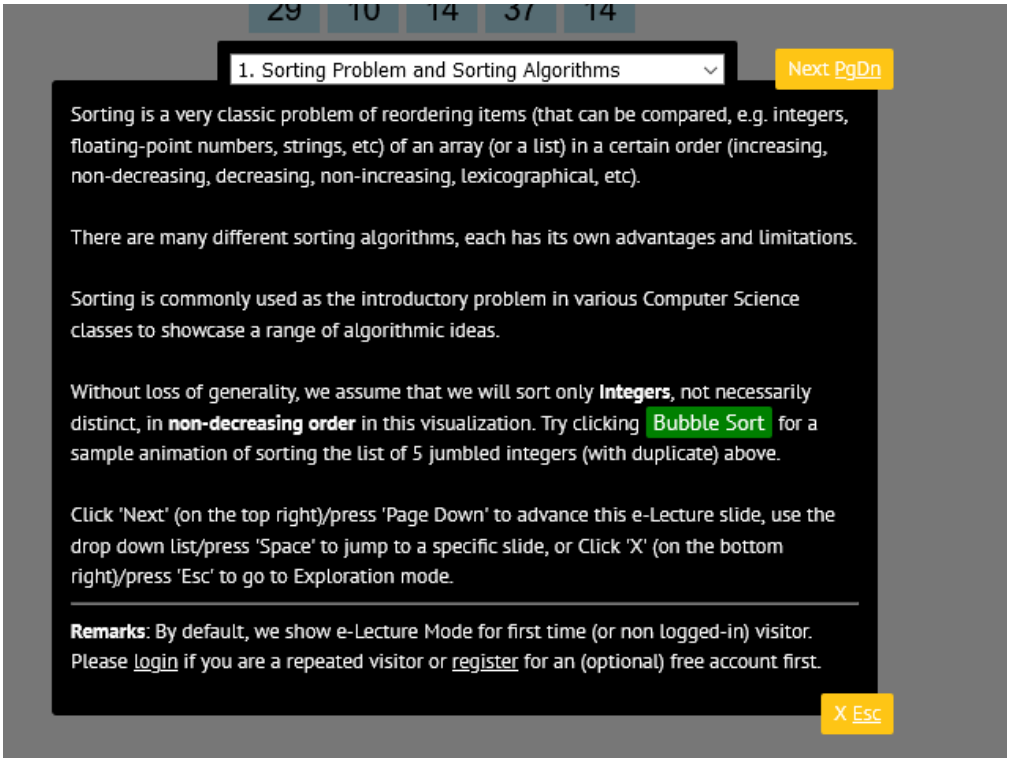

Figura 3: Texto explicación funcionamiento visualgo. Fuente: Visualgo [1]

En general, todo es demasiado cuadrado y visualmente poco atractivo. Como punto a favor de este proyecto, destacan las animaciones, las cuales están bastante conseguidas y deben ser un punto clave en el desarrollo de este trabajo.

Por otro lado, también resulta muy interesante el control de velocidad y los controles de *play*/pausa, paso adelante/atrás y volver al inicio o dirigirse al final, como se aprecia en la Figura 4. La inclusión de estos controles en nuestro trabajo podría enriquecerlo de forma muy considerable.

Figura 4: Controles visualgo. Fuente Visualgo [1]

Otro apartado interesante de esta aplicación es la inclusión de un Quiz de preguntas sobre los algoritmos seleccionados. Es configurable a nivel de dificultad, número de preguntas y tiempo límite (ver figuras 5 y 6). A la hora de plantear una posible gamificación del proyecto, considerar esta aproximación sería interesante.

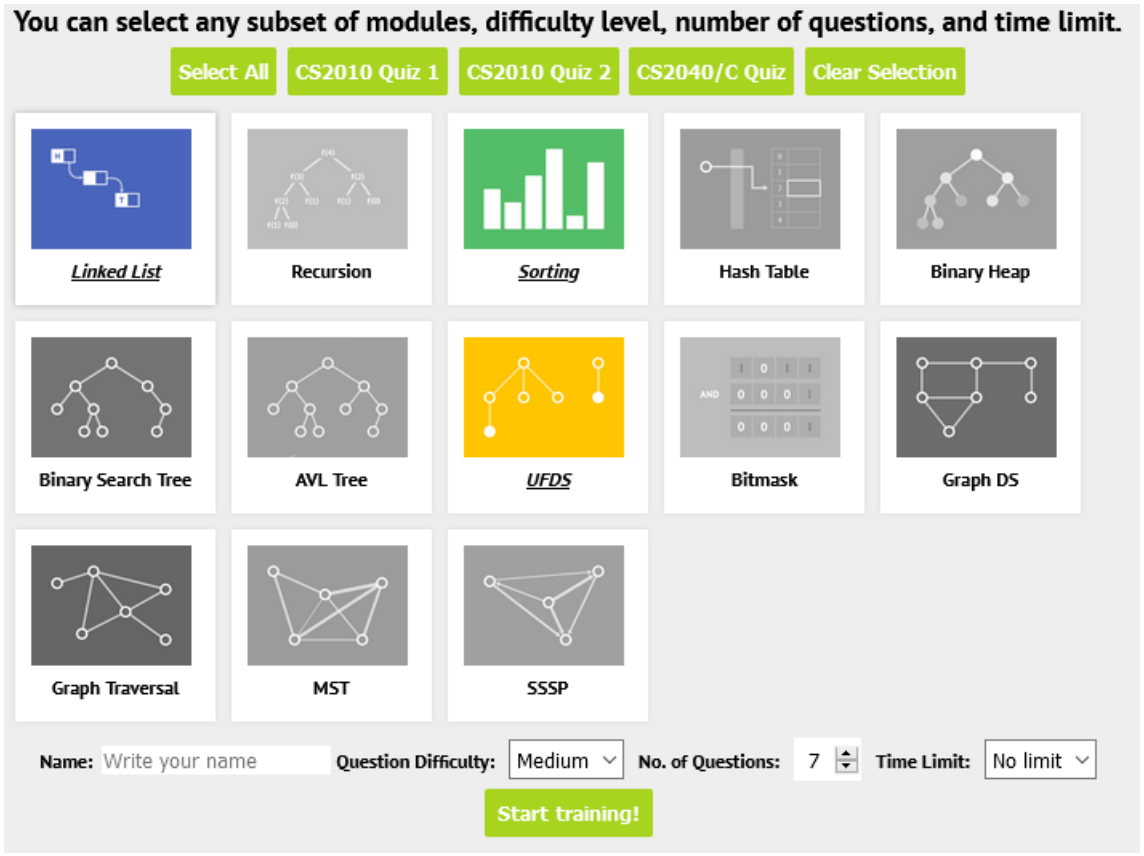

Figura 5: Quiz preguntas visualgo. Fuente: Visualgo[1]

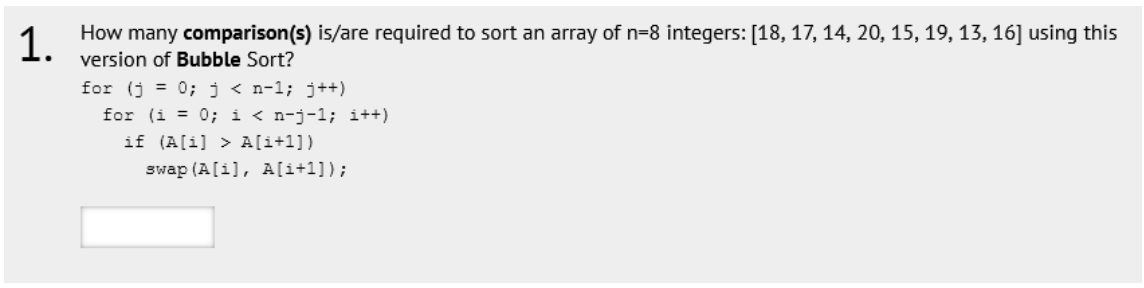

Figura 6: Pregunta quiz visualgo. Fuente: Visualgo [1]

Por último, mencionar un par de apartados que podrían ser interesantes. Por un lado, la inclusión de *tips* por las distintas pantallas, aunque definitivamente de una forma más visual y atractiva de la que propone VisuAlgo, como se puede observar en la Figura 7.

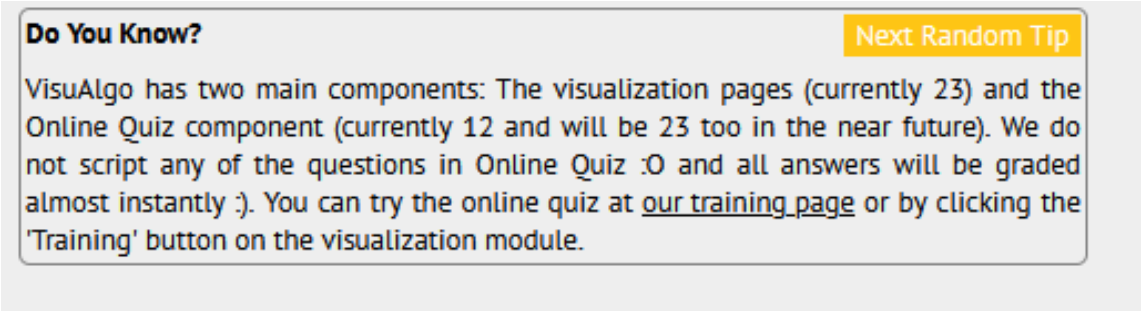

Figura 7: Consejo visualgo. Fuente Visualgo [1]

Y, por otro lado, esta aplicación es multilenguaje, característica que deberíamos implementar al menos en castellano e inglés (ver Figura 8).

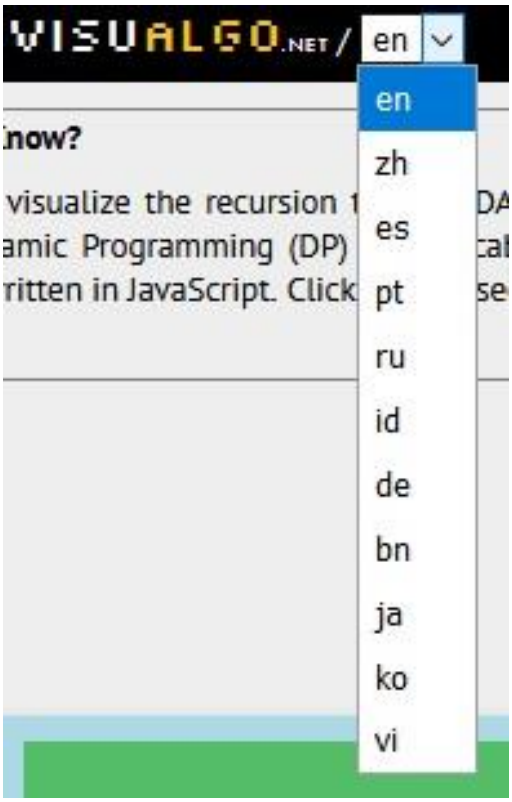

Figura 8: Idiomas disponibles visualgo. Fuente: Visualgo[1]

2.2 Algorithm Visualizer

Otra alternativa existente es la de Algorithm Visualizer [2]. Se presenta con una alternativa mucho más técnica, con un aspecto muy similar a un IDE. Su principal potencial está en la capacidad de modificar el código y ejecutarlo a tu gusto, viendo como los cambios introducidos por el usuario afectan al funcionamiento del algoritmo. Consideramos que esta característica es interesante pero quizá no lo sea tanto para la gente que empieza a estudiar el mundo de los algoritmos, la cual es claramente nuestro público objetivo principal.

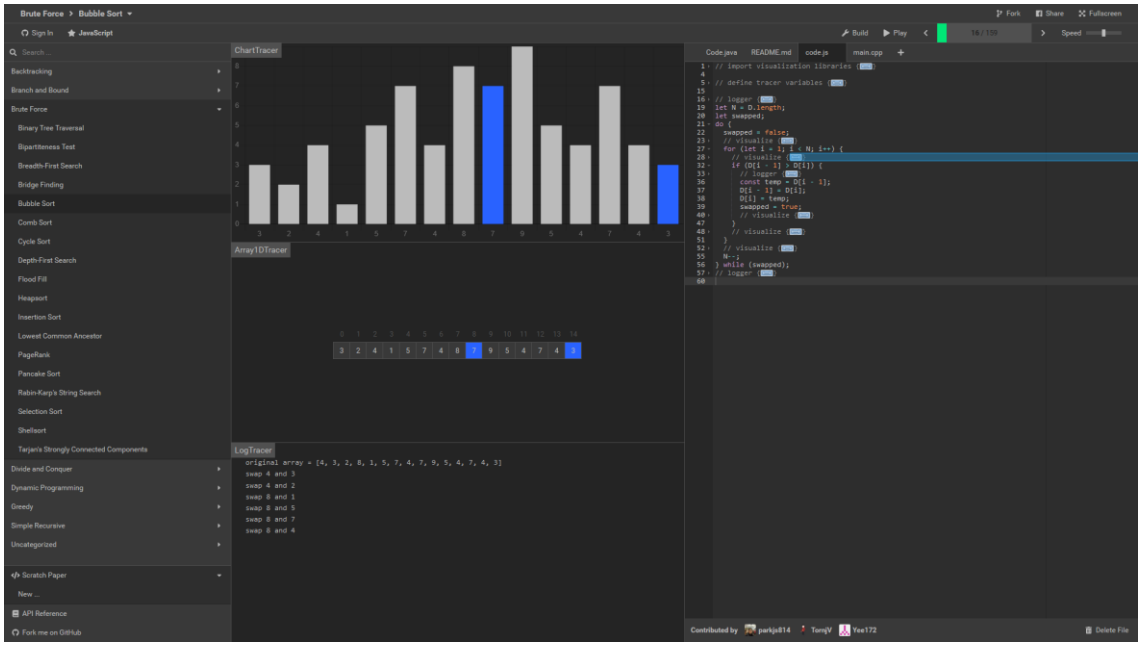

Figura 9: Interfaz principal Algorithm Visualizer. Fuente: Algorithm Visualizer [2]

Consigue aprovechar el espacio de pantalla de una forma eficiente. A la izquierda se encuentran los distintos algoritmos presentes y sus tipos. En el centro de la pantalla encontramos una traza visual, otra más clara con los datos y por último los logs que suelta el código. Y para finalizar, a la derecha se encuentra el código JavaScript modificable. Todo esto se puede apreciar en la Figura 9.

Al igual que VisuAlgo, dispone de un control de la ejecución con botón *play*/pausa, control paso por paso y regulación de velocidad. Una de sus principales carencias es la falta de animaciones en la visualización gráfica, lo cual hace que sea menos atractivo que otras propuestas.

Otra característica muy buena que remarcar es que muestra el código como si fuese un IDE, mejorando mucho su legibilidad y comprensión frente a otras alternativas. Consideramos que esta característica podría aportar bastante a nuestro proyecto.

Esta aplicación se encuentra conectada con GitHub para que los usuarios puedan aportar sus códigos. Esto es muy interesante ya que podemos ver el mismo algoritmo en distintos lenguajes de programación, para mejorar su comprensión en función del lenguaje que domine más el usuario. Esta característica lleva un problema asociado y es que sólo es posible la depuración en el código JavaScript y es necesario recargar la página para que muestre el código que has seleccionado. Por último, es posible crear un código desde cero y visualizarlo en la plataforma.

2.3 Data Structure Visualizations

Data Structure Visualizations [3] es una App web, escrita en JavaScript y usando Canvas de HTML 5. Es un proyecto de 2011 de la universidad de San Francisco creada por David Galles. Se presenta como una alternativa muy simple, con un aspecto visual web muy poco atractivo. Dispone de una buena lista de algoritmos y estructuras de datos para visualizar.

Dentro del apartado de la visualización, posee los mismos controles que el resto de sus alternativas: *play*/pausa, paso atrás/adelante, ir al inicio/final y control de la velocidad de ejecución. Es posible cambiar el tamaño del Canvas y no tiene visualización de código. Podemos apreciar esto en la Figura 10.

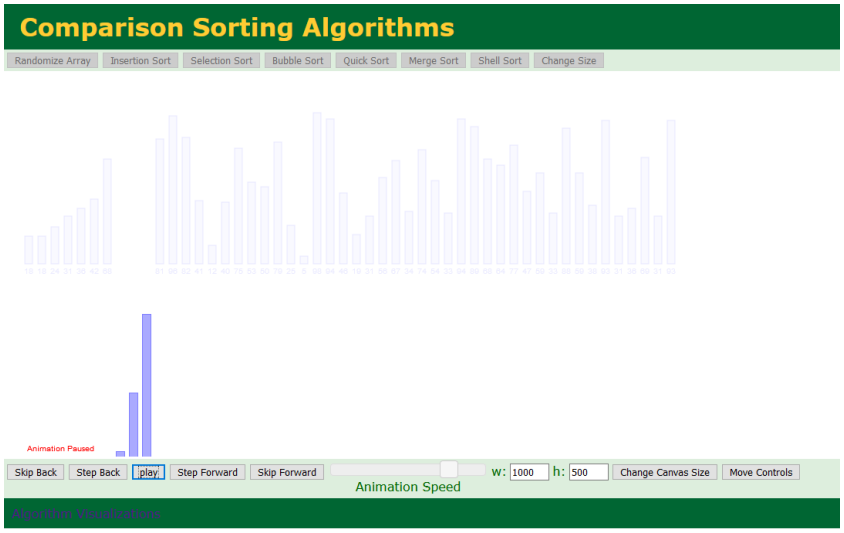

Figura 10: Visualización algoritmos ordenación Data Structure Visualizations. Fuente: Data Structure Visualizations [3]

Visualmente es muy limitado, aunque si dispone de animaciones en los gráficos. Otro punto que no juega en su favor es la gran cantidad de elementos a ordenar, lo cual dificulta su comprensión para el usuario. Con muchos menos elementos sería más claro. Un buen punto para nuestro trabajo sería la posibilidad de configurar el número de elementos, dentro de unos límites para no complicar la comprensión del funcionamiento del algoritmo al usuario.

2.4 Algomation

Algomation [4] se presenta como una alternativa en la cual podemos visualizar, crear y compartir cualquier tipo de algoritmo. Por lo tanto, se fundamenta en la idea de que todos los algoritmos en la plataforma son públicos y cualquier usuario registrado puede hacer su aportación. Para ello nos presenta un buscador como página principal. Se encuentra desarrollado en NodeJS/Express y usa una base de datos Mongo.

La idea de que su entrada principal sea la de un buscador debido a la gran cantidad de aportes esperados por los usuarios es interesante, pero consideramos que se aleja de nuestra propuesta.

Visualmente, la página principal es atractiva y simple. Presenta un buscador y una lista de los algoritmos más visitados. Al seleccionar un algoritmo, nos presenta en primer lugar una información básica del aporte, que incluye el nombre, descripción, autor y resumen del resultado final.

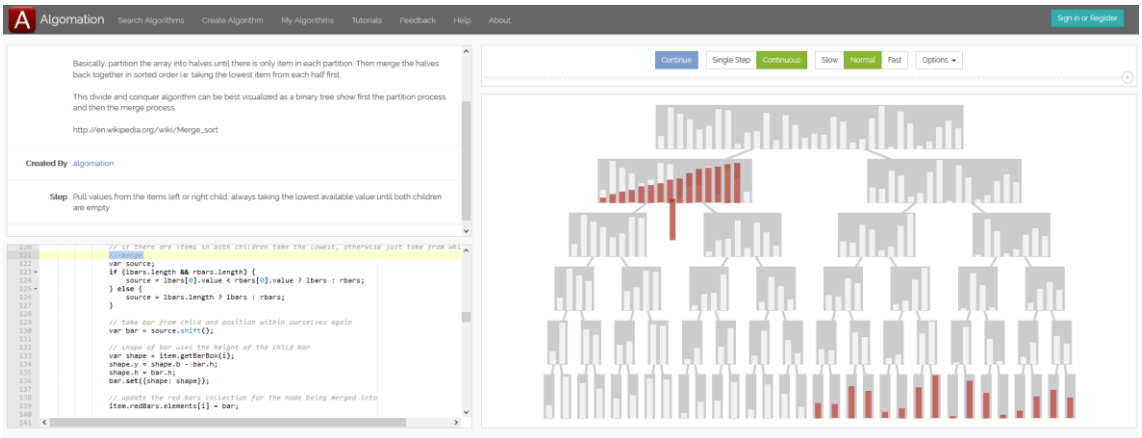

Figura 11: Visualización algoritmo ordenación Aglomotion. Fuente: Algomation [4]

Por debajo de esto nos encontramos el código resaltando por el paso que va la ejecución. Este código también se presenta como si fuese un IDE, similar al de Algorithm Visualizer, pero sin el resalte de colores para mejorar la comprensión y sin poder editar el mismo.
En la parte derecha de la pantalla se nos presenta la visualización como tal, empleando animaciones de una forma bastante limpia y atractiva. Consideramos que el añadir un valor a cada barra mejoraría bastante la comprensión a la hora de ordenar los elementos.

Un elemento diferenciador de esta alternativa es el control de la ejecución, sobre todo la línea de tiempo que nos permite seleccionar el paso deseado. Además, incluye botón de *play*, poder seleccionar si queremos ir paso por paso o de forma continua, control de velocidad y una lista de opciones. Esta lista de opciones incluye: crear una nueva rama de desarrollo a partir de ese algoritmo, compartirlo, obtener el script, reiniciar el algoritmo y poder ver más aportaciones del autor (ver Figura 11).

Por último, en la parte inferior se presenta una sección de comentarios que debido a su gran enfoque social consideramos que es acertada. Podemos apreciar esto en la Figura 12.

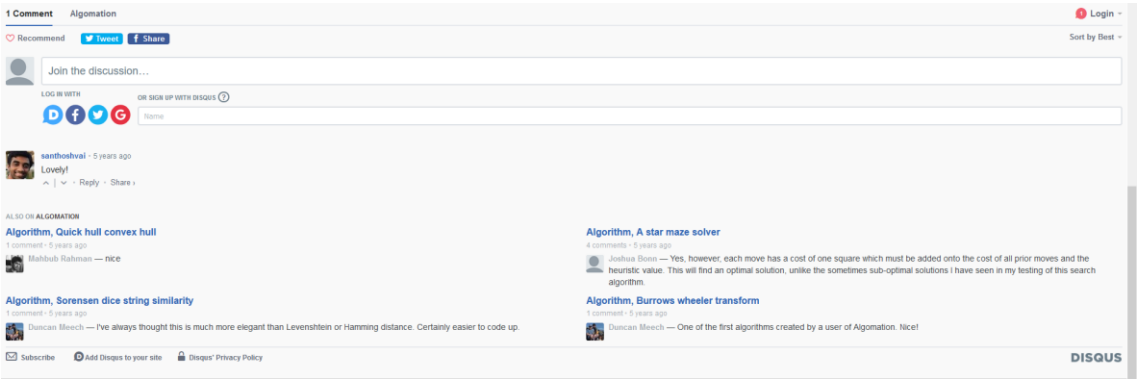

Figura 12: Apartado Social Algomotion. Fuente Algomation [4]

Finalmente, la plataforma cuenta con una sección de tutoriales para ayudar a crear aportes a la aplicación, una sección de ayuda con apartados para todo lo relacionado con la aplicación y una sección para proporcionar *feedback* al creador.

2.5 Videos Youtube

El principal problema que presentan este tipo de videos es el gran número de elementos a ordenar y la velocidad de la ejecución, por la cual apenas permite entender cómo están funcionando los distintos tipos de algoritmos. Podemos apreciar esto en la Figura 13.

Figura 13: 15 sorting Algorithms in 6 minutes [5]

Capítulo 3 - Objetivos

3.1 Apoyo académico

El objetivo principal de la aplicación, desde su concepción, ha sido el de una herramienta de apoyo a los estudiantes de las distintas asignaturas de algoritmos, ya sea de la universidad o de cualquier otro tipo de formación. Consideramos que contar con una herramienta simple, pero eficaz, podría ayudar de manera muy positiva a la comprensión del funcionamiento de distintos algoritmos.

3.2 Simple e intuitiva

Nuestra principal baza a la hora de diferenciarnos de las alternativas es buscar que la herramienta sea lo más sencilla e intuitiva posible. Nuestro objetivo aquí es el de no desbordar al usuario con demasiada información y que la presencia de elementos en la pantalla esté totalmente justificada para facilitar un aprendizaje eficaz. Asimismo, la navegación debe ser lo más simple posible para no desorientar al usuario y que sepa en cada momento a donde dirigirse

3.3 Herramienta universal

Aparte de ser una herramienta de apoyo académico, consideramos que la aplicación debe ser lo más accesible posible a cualquier tipo de usuario, sean cuales sean sus razones para utilizarla. Es por ello por lo que debemos buscar la mejor plataforma, con el mínimo de requisitos, para lograr el máximo alcance. En el logro de este objetivo es clave la optimización de la aplicación.

Capítulo 4 - Descripción Informática

4.1 Herramientas Utilizadas

4.1.1 Unity

La elección de Unity [6] como herramienta viene por tres aspectos principales. El primero, el uso de lenguaje C#, sencillo, potente y con muy buena documentación. En segundo lugar, la experiencia en el uso de la herramienta en varias asignaturas de la carrera. Y por último en tercer lugar, las grandes posibilidades de exportación a muchas plataformas de forma muy sencilla. La versión empleada para desarrollar la aplicación es la 2019.2.19f1.

Consideramos que la elección de esta herramienta es un total acierto debido a las posibilidades que nos aporta de una forma muy sencilla. La aplicación consistirá principalmente en una interfaz gráfica, sin mucho contenido 2D o 3D por detrás, ya que sobrecargaría la web sin necesidad. El funcionamiento básico de Unity se basa en la construcción de escenas, en las que introducimos una serie de objetos, los cuales llevan asociados unos determinados componentes.

Un ejemplo de escena sencilla en Unity sería el de un objeto bola y otro objeto suelo, en la cual podríamos manejar el movimiento de esta. Estos objetos llevarían asociados componentes de tipo Malla 3D para poder visualizarlos. A su vez, se les añadirían componentes para manejar las interacciones físicas entre ellos, lo cual ya se encuentra desarrollado por Unity y simplifica mucho el desarrollo. Por último, un componente de tipo *script* para controlar el movimiento por teclado. Se podrían añadir elementos extras como un objeto de interfaz gráfica, que mostrase por ejemplo la dirección o la velocidad a la que va la bola. En la siguiente sección trataremos en detalle cómo se organiza un proyecto de Unity

4.1.1.1 Organización proyecto Unity

Para ayudar a la comprensión del desarrollo del proyecto, en primer lugar, introduciremos como se organiza un proyecto de Unity. El proyecto se divide en escenas, las cuales contienen *game objects* y estos a su vez distintos componentes.

4.1.1.1.1 Escenas

Dentro de un videojuego cada una de estas escenas podría ser un nivel distinto, con su modelado 3D, sus personajes, *scripts* que controlen a estos personajes, etc. Las escenas nos permiten tener cargado solo lo que nos interesa en ese momento, es por ello por lo que es muy importante hacer un buen diseño. En lo que se refiere a nuestro proyecto, hemos decidido organizarlo creando una escena individual para cada algoritmo y por otro lado una escena que gestione el menú principal. El resultado se muestra en la Figura 14.

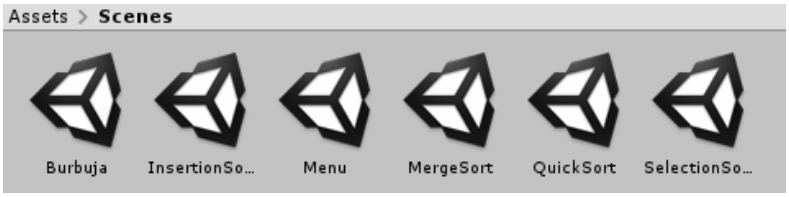

Figura 14: Escenas del proyecto V-Algo

4.1.1.1.2 Game Objects

Los *game objects* en Unity son los objetos que se encuentran en la escena, ya sean visibles o no y son contenedores de componentes, los cuales definen al *game object.* La única característica común a todos ellos es que siempre llevan asociados un componente *transform*, que define su posición, rotación y escala dentro de la escena. A partir de esa base, se pueden añadir todos los componentes que sean necesarios. Algunos ejemplos de *game objects* dentro de nuestro proyecto pueden ser la cámara, la cual incluirá los componentes necesarios para su correcto funcionamiento o por otro lado el *LanguageManager*, el cual simplemente contiene un componente script, encargado de traducir los componentes de texto al idioma seleccionado por el usuario (ver Figura 15).

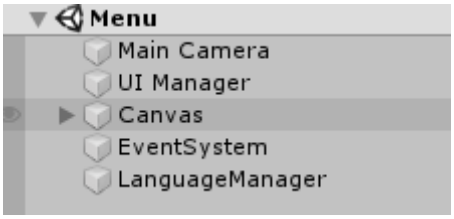

Figura 15:Game Objects de la escena del menu de V-Algo.

4.1.1.1.3 *Prefabs*

Los *prefabs* dentro de Unity son *game objects* con distintos componentes asociados y con valores establecidos por nosotros de los cuales guardamos una copia exacta. Esta característica dentro de Unity es muy útil, ya que nos permitirá instanciar tantas copias iguales del mismo *game object* como queramos, puesto que actúa como una plantilla. De la misma forma, si hacemos un cambio en el *prefab*, este cambio se aplicará a todas las instancias de este.

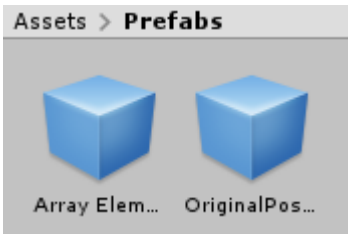

Figura 16: Prefabs del proyecto V-Algo

Dentro de nuestro proyecto, por ejemplo, se encuentra el *prefab Array Element*, como podemos ver en la Figura 16. Este *prefab* es utilizado en todos los algoritmos para representar cada uno de los elementos del array. Es por esto una pieza esencial dentro del proyecto y se describirá en profundidad en la sección 4.2.1

4.1.1.1.4 Componentes

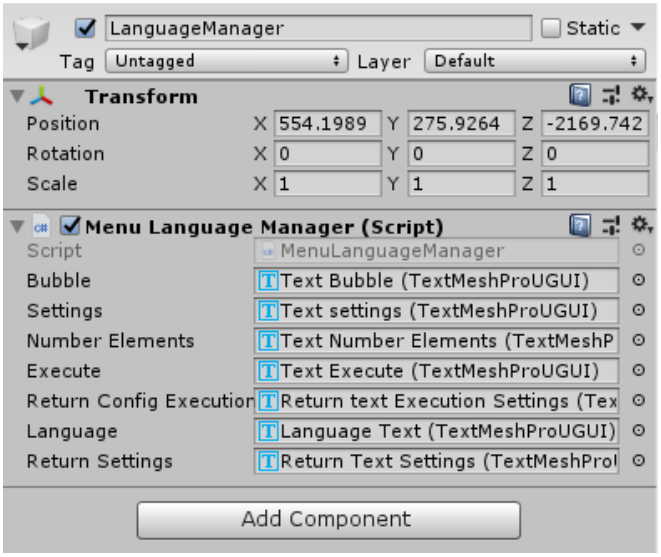

Figura 17: Componentes del Game Object LanguageManager.

Como hemos comentado antes, los componentes definen a los *game objects* en todos los aspectos. Por ejemplo, si a un *game object* vacío, solamente con las propiedades de su *transform*, le añadimos una malla 3D de un cuadrado, tendremos en la escena un cuadrado determinado por los valores de la *transform* del objeto.

Por ejemplo, en la Figura 17, podemos ver el *game object Language Manager*, el cual lleva asociado 2 componentes: la transform y un script de código C#. Los distintos elementos que se ven dentro del componente del script son las propiedades públicas del código, que se establecen directamente desde el editor.

4.1.1.1.5 Assets

En el explorador de nuestro proyecto encontraremos siempre una carpeta raíz llamada *Assets*. Esta carpeta contendrá todo lo necesario para ejecutar nuestro proyecto con normalidad.

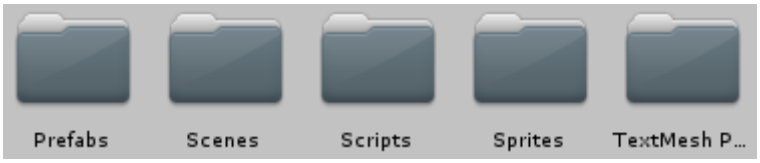

Figura 18: Assets del proyecto V-Algo.

En la Figura 18 se pueden ver las carpetas presentes en el proyecto: *prefabs*, escenas, *scripts*, *sprites y TextMesh Pro*. La carpeta de *Text Mesh Pro* es un paquete con componentes de texto que proporcionan una calidad mucho mayor respecto al componente de texto nativo de Unity.

4.1.1.2 Uso de interfaces gráficas y no mallas 2D o 3D

Nuestro trabajo se centra en el desarrollo de objetos de tipo interfaz gráfica, ayudándonos de los que ya tiene implementados Unity. Para la visualización de algoritmos no usamos objetos 3D con mallas, ya que no sería la solución eficiente y se sacrificaría la optimización. En todas las escenas hay presente un objeto *Canvas* y todos los objetos visuales serán hijos de él y de tipo interfaz gráfica.

Por ejemplo, a la hora de visualizar un algoritmo de ordenación, representaremos el contenido del array a través de barritas, que serán objetos de tipo interfaz gráfica. Estos objetos serán modificados para ajustar su altura en función del número que representen, el cual también será mostrado en la barra, para ayudar a su comprensión. Podemos apreciar esto en la Figura 19.

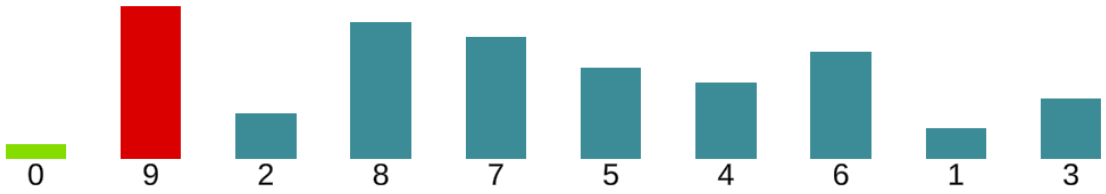

Figura 19: Representación Array V-Algo. Fuente: Elaboración propia

4.1.1.3 Manejo de escenas

Según hemos hablado de cómo funciona Unity en torno a las escenas, es necesario definir el número de ellas que forman nuestra aplicación. En una primera aproximación pensamos en una escena para el menú y la navegación por él, y una escena para cada tipo de algoritmos desarrollado, ya que cada uno tendrá sus peculiaridades a la hora de realizar la visualización. Más adelante podríamos pensar en subdividir el menú en varias escenas, en función de las pantallas que necesitemos, aunque el cambio entre escenas podría ser más ineficiente que cambiar el contenido de una escena única.

4.2 Bases para el resto de los algoritmos

El desarrollo del algoritmo de la burbuja fue el más laborioso y largo de todos debido a que fue el primero en ser desarrollado. Con el desarrollo de este algoritmo se pretendía crear una base común para facilitar el desarrollo del resto.

4.2.1 Prefab Elementos del array

Es en este proceso cuando nace una pieza esencial del proyecto: el *game object* que representa a cada uno de los elementos gráficos del array. Queríamos que fuese algo simple y que el usuario pudiese ver fácilmente la diferencia entre 2 de estos elementos asociados a distintos números dentro del array.

Al ser un elemento que estaría presente en todos los algoritmos y varias veces dentro de ellos, era necesario crear un *prefab*, una plantilla de las cuales hemos hablado en la sección 4.1.1.1.3. Esto nos permitiría instanciar copias exactas con mucha facilidad.

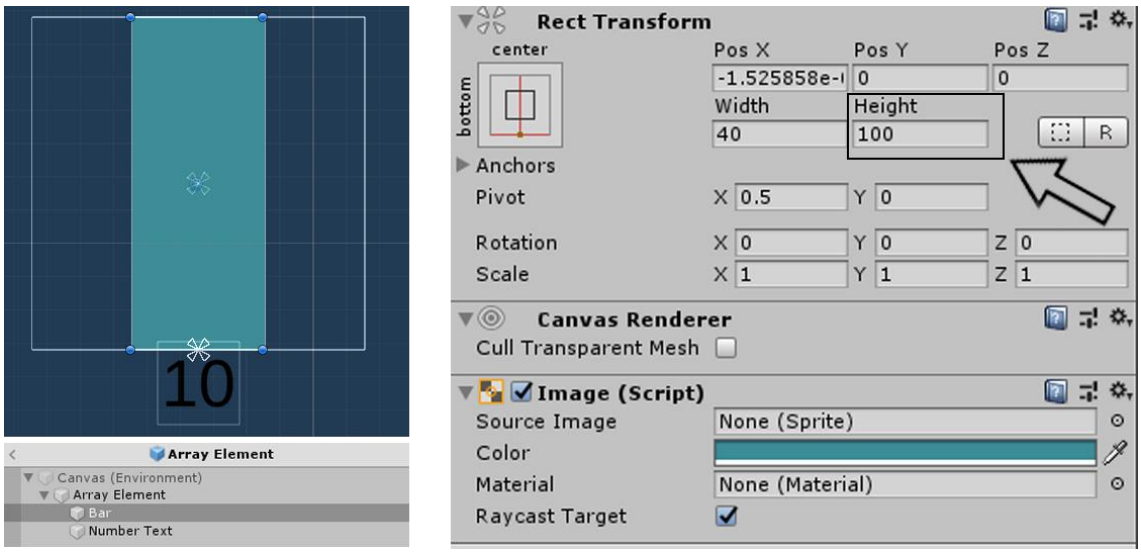

Figura 20: Prefab Array Element.

Por ello empezamos a diseñar un *game object* que se compone fundamentalmente de dos partes. En primer lugar, un número en la parte inferior, que representa en cada instancia de este *prefab* a un número distinto de los incluidos en el array a ordenar. En segundo lugar, una barra justo encima del número, cuya altura debe ser ajustable mediante código para poder definirla en función del número asignado a la instancia. Podemos observar esto en la Figura 20.

4.2.2 Creación Canvas

Como ya hemos hablado con anterioridad, nuestra intención con esta aplicación es que se manejase únicamente por componentes de tipo UI. El uso de componentes basados en mallas podría afectar al rendimiento de la aplicación innecesariamente. Es por ello por lo que todos nuestros objetos visibles estarán basados en componentes de interfaz gráfica de Unity, como imágenes, botones, textos, etc.

Todos estos objetos se organizan como hijos de un *game object Canvas* con un componente *Canvas* dentro de él. Este objeto será el encargado de renderizar nuestros objetos empezando por el primer hijo hasta el último y de igual forma en las jerarquías inferiores. Debemos tener esto en cuenta a la hora de poner imágenes de fondo y que elementos superponen a otros.

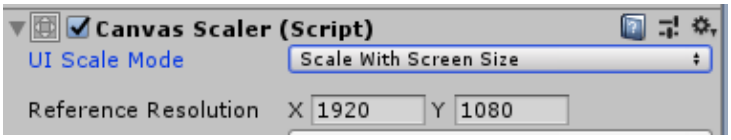

Figura 21: Ajustes Canvas Scaler.

Es importante el ajuste de *UI Scale Mode* de *Canvas Scaler* a *Scale With Screen Size* (ver Figura 21). Esto permite que todos los elementos dentro del *canvas* escalen si aumentamos el tamaño de la ventana o pasamos al modo pantalla completa.

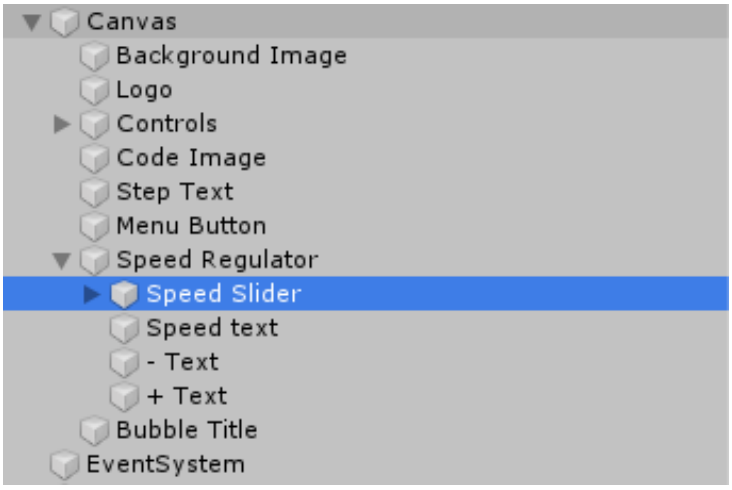

Figura 22: Canvas de la escena del algoritmo burbuja.

Como podemos ver en la Figura 22, la imagen de fondo debe ser siempre la primera hija, para que se renderice detrás de todo el resto de los elementos. A continuación, pasaremos a hablar de los elementos del *canvas* comunes a todos los algoritmos. El *canvas* va acompañado siempre del objeto EventSystem para ayudarle al sistema de mensajes.

4.2.2.1 *Background Image* / Imagen Fondo

Es la imagen que se encuentra al fondo de todos los elementos UI. Es importante que configuremos los puntos de anclaje para que se ajusten siempre al tamaño de pantalla y la imagen escale y se redimensione si la ventana cambia.

4.2.2.2 Logo

Simplemente es un objeto que contiene un componente *Image*, cuya *source image* es el logo del proyecto, como se puede observar en la Figura 23.

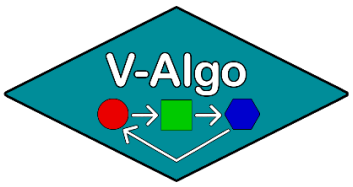

Figura 23: Logo V-Algo.

4.2.2.3 *Controls* / Controles

El objeto *controls* es un objeto padre que contiene un hijo botón por cada uno de los controles añadidos al control de la ejecución del algoritmo.

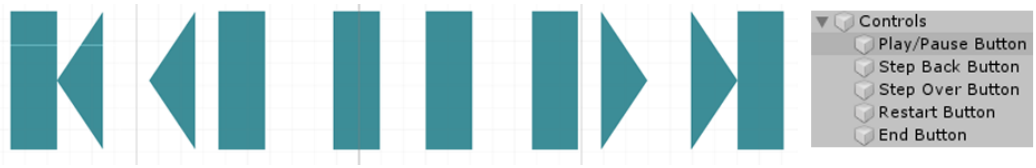

Figura 24: Controles de ejecución.

Cada uno de estos botones lleva asociado un componente imagen que es lo que vemos y por otro lado un componente botón para programar su comportamiento por código. Los podemos observar en la Figura 24.

4.2.2.3.1 Comportamiento botones de control

Los botones de control nos permiten controlar la ejecución del algoritmo. Esta ejecución dispone de 2 modos: automático o paso por paso. Una vez entramos al algoritmo este siempre empieza a ejecutar en automático. Si queremos usar los botones de paso atrás/adelante, deberemos pausar la ejecución pulsando el botón de *play*/pausa. Los botones de reiniciar la ejecución o ir al final de esta siempre están disponibles para su uso, independientemente del modo en el que nos encontremos. Al pulsar uno de estos 2 botones, la ejecución siempre se detendrá y deberemos pulsar el botón de *play* si queremos que vuelva al modo automático.

4.2.2.4 Code Image / Imagen del código

Este objeto contiene un componente imagen cuyo *source* ira cambiando en función de la línea de código que se encuentre en ejecución. Es por esto por lo que ha sido necesario crear una imagen con el código limpio y luego una por cada tipo de estado/instrucción a remarcar en la ejecución en cada uno de los algoritmos, como se puede apreciar en la Figura 25.

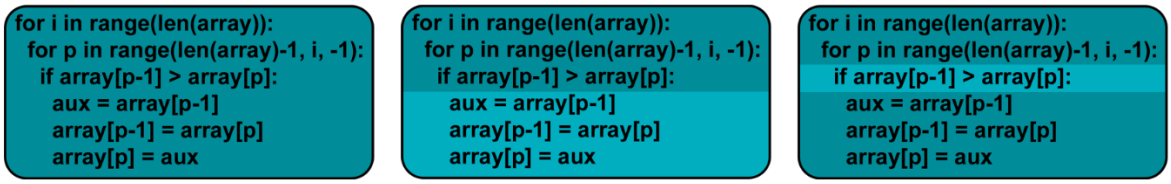

Figura 25: Imágenes del código del algoritmo burbuja.

4.2.2.5 Step Text / Texto de paso

Es simplemente un objeto que contiene un componente *UI Text Mesh Pro*. Se encargará de mostrar el paso por el que va la ejecución y será traducido de forma automática por el script al idioma que haya seleccionado el usuario, como podemos ver en la Figura 26.

Figura 26: Texto de paso.

4.2.2.6 Menu Button / Botón Menú

Este botón nos permitirá volver al menú desde cada uno de los algoritmos. Lo podemos apreciar en la Figura 27

Figura 27: Botón menú.

4.2.2.7 Speed Regulator / Regulador de velocidad

Es el objeto encargado de regular la velocidad de ejecución. No solo afectará al modo automático, sino que también regulará la velocidad a la que se producen los movimientos de los elementos en el modo paso por paso.

Figura 28: Regulador de velocidad.

Como podemos observar en la Figura 28, está compuesto por varios elementos. En primer lugar, por 2 imágenes con los símbolos + y -. En segundo lugar, contiene un texto con la cadena "velocidad," que será traducido al idioma seleccionado por el usuario automáticamente. Y, por último, el componente verdaderamente encargado de gestionar la velocidad, un *slider*. Este componente ya viene preparado por Unity y solamente es necesario configurar los valores deseados como valor máximo, mínimo y con el que empezará la ejecución.

Como en el proyecto entendemos la velocidad como el tiempo de parada entre cada paso, fue necesario voltear el *slider* para que los valores más altos tendiesen hacia la izquierda - (los más lentos) y los más pequeños (los más rápidos), hacia la derecha +.

4.2.3 Language Manager / Gestor de lenguaje

Este objeto se encuentra presente en cada una de las escenas, y dentro de cada una de ellas contiene un script diferente. Cada uno de estos scripts se encarga de traducir aquellos textos que varían dependiendo del idioma que haya seleccionado el usuario en el menú.

4.2.4 Main Camera / Cámara principal

De la cámara simplemente remarcar que se encuentra por defecto en todas las escenas. Es la encargada de renderizar todo lo presente en la escena de forma 2D o 3D. Al estar basado todo nuestro proyecto en elementos UI, no es realmente necesaria en ningún momento, aunque se ha dejado presente debido a que si la eliminamos el editor mostrará siempre un error.

4.2.5 Función MoveToPosition

Para poder realizar los movimientos de los elementos gráficos del array se nos presentaban 2 posibles opciones: a través de animaciones o mediante código. Unity posee un motor de animaciones con muchas posibilidades, pero descartamos esta opción debido a tener escasa experiencia trabajando con él. Por consiguiente, nos decantamos por la opción de código debido a nuestro mayor conocimiento en la misma, al haber sido trabajada a lo largo de diversas asignaturas del grado.

El movimiento de los elementos gráficos del array está presente en la visualización de todos los algoritmos del proyecto. Es por tanto un aspecto clave y debe estar sustentado en una buena función de movimiento, que actúe de forma correcta en distintas situaciones. Para apoyarnos en el desarrollo de este código, hemos usado la función `Lerp` de Unity, la cual permite interpolar entre dos posiciones Vector3 en base a un parámetro *float* t, el cual debe estar entre 0 y 1. Cuando este parámetro tiene valor 0, la función devuelve la posición original y cuando es 1, la de destino. Normalizando la duración del desplazamiento entre estos valores conseguimos la función deseada, la cual podemos ver en la Figura 29.

```
//Function that allows animate the permutations of the array
6 referencias
private IEnumerator MoveToPosition(GameObject initialObject, Vector3 endPosition, float timeToMove)
{
    float time = 0f;
    Vector3 startPosition = initialObject.transform.position;

    while (time < 1)
    {
        time += Time.deltaTime / timeToMove;
        initialObject.transform.position = Vector3.Lerp(startPosition, endPosition, time);
        yield return null;
    }
}
```

Figura 29: Función MoveToPosition.

Otro aspecto que destacar de esta función es que obligatoriamente tiene que ser una corrutina. Esto es debido a que necesitamos que le asigne un nuevo valor a la posición del objeto en cada *frame.* Con la llamada `yield return null`, le estamos diciendo a Unity que la siguiente iteración del bucle se ejecute en el siguiente *frame*. Si esta función no fuese una corrutina, la asignación de nuevas posiciones al objeto sería tan rápida que daría el efecto de un teletransporte y no una animación de movimiento.

4.3 Cambio de enfoque a estados

Nuestra primera aproximación a la hora de visualizar la ejecución del algoritmo fue la de ejecutar el código de este solo 1 vez y entre instrucciones de esta ejecución ir realizando paradas o lo que fuese necesario. Este enfoque es correcto si solo permites al usuario avanzar en la ejecución, ya sea de forma automática o paso por paso.

El conflicto surge cuando se desea incluir que el usuario pueda volver al inicio de la ejecución o peor aún, que retroceda paso por paso en la misma. Buscando posibles soluciones a este problema surgió la idea de intentar revertir las llamadas a la pila para volver atrás en la ejecución. Esta opción finalmente fue descartada al considerarla demasiado compleja, especialmente en algoritmos recursivos como *Merge Sort* o *Quick Sort.*

Pensando en las funcionalidades deseadas para la visualización del algoritmo, como retroceder, volver al principio o avanzar hasta el final, surgió una idea con un enfoque totalmente distinto. Comprendimos que, para conseguir un control total sobre la ejecución, esta debería estar basada en una lista de estados y reproducir la visualización a través de esta.

Por lo tanto, decidimos ejecutar el algoritmo de forma completa antes de empezar la reproducción y guardar entre líneas de esa ejecución una lista con los estados que nos interesa reproducir. Estos estados están representados por una clase propia para cada algoritmo y contienen las propiedades necesarias para su correcta reproducción. La ejecución del algoritmo y el almacenamiento de los estados en una lista se produce en la función `ExecuteAndCreateStates` dentro del script que controla cada algoritmo.

4.4 Funciones Start() y Update()

Consideramos importante el hablar sobre dos funciones proporcionadas por Unity y que son indispensables en el correcto funcionamiento del proyecto. De hecho, son las únicas que siempre vienen escritas al crear un script de Unity.

En primer lugar, la función `Start()`. Todo lo que se encuentre dentro de esta función será ejecutado antes de cargar la escena. Debido a esto, la aprovechamos para hacer la traducción al lenguaje deseado, establecer las variables y realizar las acciones necesarias relacionadas con el array a ordenar. Estas acciones sobre el array son: originarlo de forma aleatoria, crear los estados en base a la ejecución del algoritmo sobre el mismo e instanciarlo de forma gráfica.

En segundo lugar, la función `Update()`. Esta función es llamada automáticamente en cada *frame*. Por lo tanto, será la encargada de gestionar la reproducción de la ejecución, tanto en el modo automático como en el modo paso por paso, añadiendo la lógica asociada a los botones de paso atrás/adelante.

4.5 Uso de corrutinas

A la hora de controlar la velocidad de la ejecución se nos presentaba la necesidad de tener alguna función que esperase un tiempo determinado sin parar por completo el hilo principal de ejecución. Unity tiene una herramienta para estos propósitos y son las corrutinas. Estas, en combinación con variables de control, nos han permitido mantener el tiempo de espera entre cada paso y permitir que los movimientos de los elementos del array terminen antes de seguir con el siguiente paso.

4.6 Diagrama de clases

En la figura 30 se muestra el diagrama de clases del proyecto. Cada uno de los algoritmos tiene asociado 3 *scripts*. El general, con el nombre del algoritmo, se encarga de controlar toda la ejecución de este. Para que esto funcione de forma correcta, es necesario crear una lista de estados, los cuales están representados por una clase propia en cada algoritmo. En segundo lugar, hay un script Language Manager por cada algoritmo, encargado de traducir todos los elementos de texto de cada escena.

Por otro lado, se encuentran los scripts asociados al menú. Hay uno por cada submenú y están encargados de controlar aspectos específicos de cada uno de ellos. Además, hay un *Language Manager* que traduce los textos de todos los menús y un *UIManager*, el cual se ocupa de aspectos generales del menú y su navegación, desactivando o activando los distintos submenús.

Como se puede apreciar en la Figura 30, todas las clases, menos las que representan los estados, heredan de la clase de *Unity Mono Behaviour*, que a su vez hereda de *Behaviour*, *Component* y *Object*. Estas clases son propias del editor y son la base de la que todos los scripts de Unity deben heredar, ya que contienen todas las propiedades y funciones necesarias para desarrollar nuestros proyectos en este editor. Algunos ejemplos de estas funciones son las de `Start() y Update()`, de las cuales hemos hablado en la sección 4.4.

Hemos incluido el archivo del diagrama de clases en la carpeta de *Scripts* del proyecto. Este archivo se puede abrir con Visual Studio y ver más en profundidad todos los detalles de las clases y sus relaciones. Para abrirlo es necesario añadir la extensión de Visual a través del instalador.

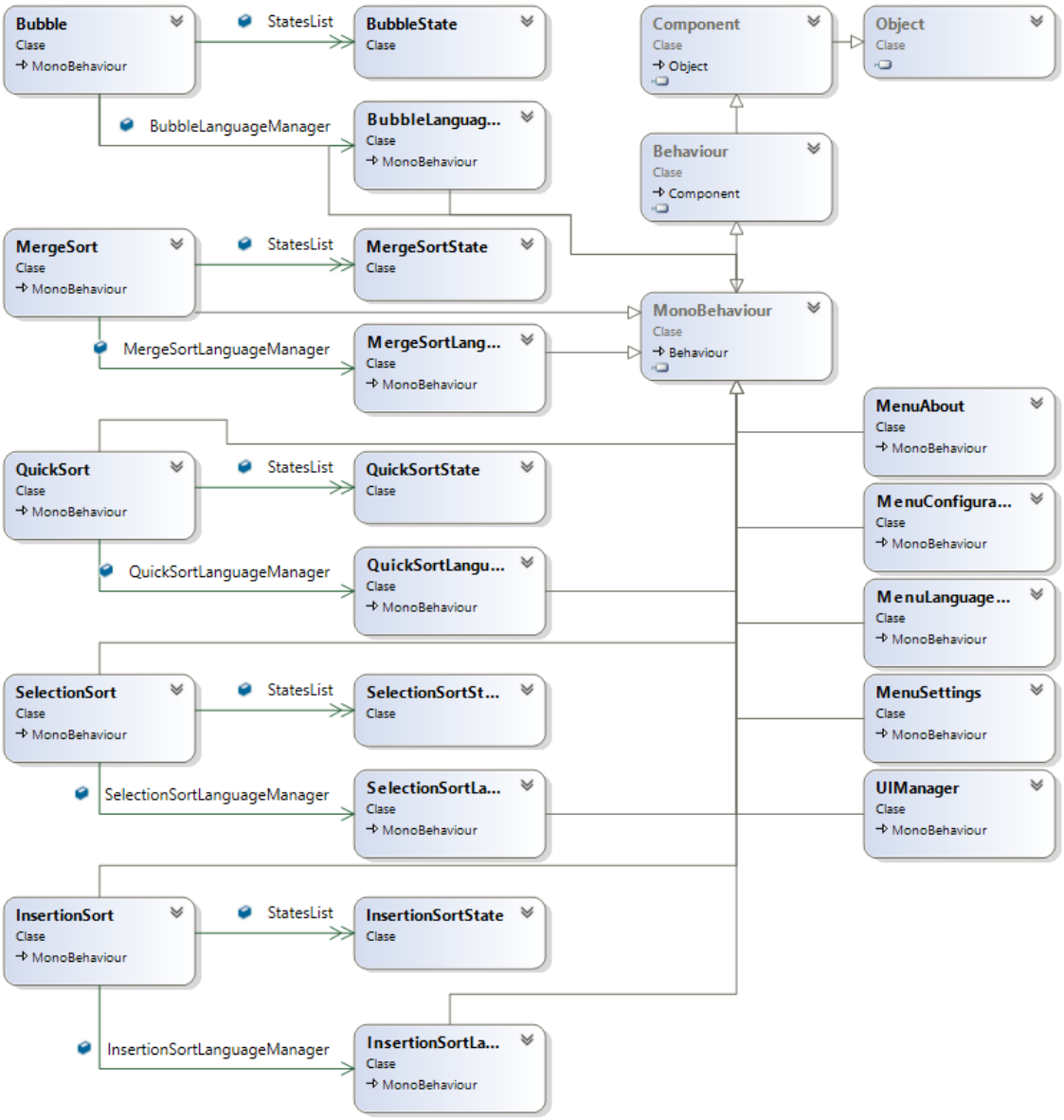

Figura 30: Diagrama clases del proyecto

4.7 Desarrollo algoritmo burbuja

Estados Burbuja

El algoritmo de la burbuja es el más sencillo de todos, pero a la vez el menos eficiente. Es por ello por lo que solo tiene 2 estados a remarcar: IF y SWAP de elementos

If - 0

Comprueba si el elemento de la izquierda es mayor que el de la derecha. Se remarcan los elementos comparados y esa zona del código, como se aprecia en la Figura 31.

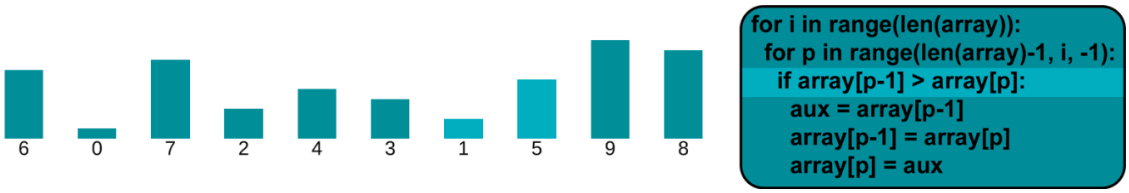

Figura 31:Estado if burbuja.

Swap - 1

Intercambia los elementos de la izquierda con el de la derecha si el IF ha sido correcto. Se remarcan los elementos, la zona de código, se intercambian los elementos y se hacen los cambios necesarios en la lista de los elementos gráficos del array para que correspondan con el estado actual, como se puede ver en la Figura 32.

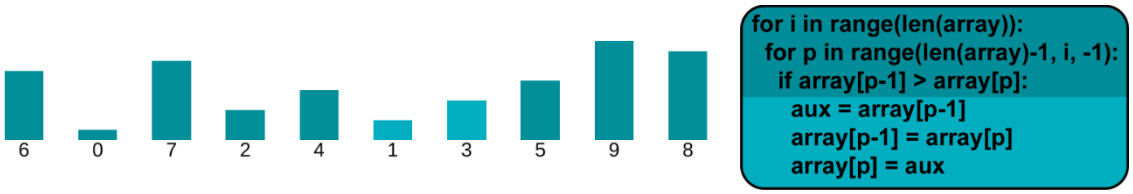

Figura 32:Estado swap burbuja.

4.8 Desarrollo algoritmo Merge Sort

El segundo algoritmo desarrollado fue el de *Merge Sort*. Este algoritmo tuvo un desarrollo más complejo que el de burbuja, debido principalmente a ser recursivo, entre otras razones.

Funcionamiento por colores

Debido a que este algoritmo funciona de forma recursiva por mitades, de menor a mayor tamaño, se ha decidido el dar un color aleatorio a cada elemento al comienzo de la ejecución. Esto se realiza para que sea más claro en qué mitad está trabajando y cuáles de estas mitades ya se encuentran ordenadas, ya que se agruparán por un mismo color.

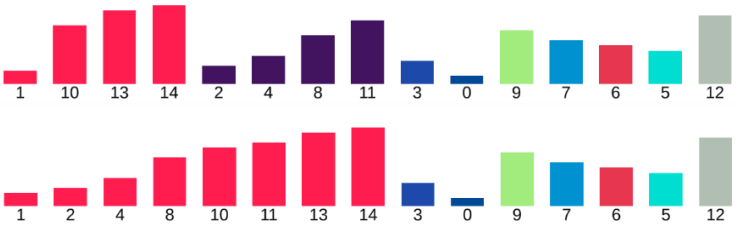

Figura 33: Merge Sort mitades ordenadas por colores.

Como se puede apreciar en la Figura 33, las 2 mitades de la izquierda ya se encuentran ordenadas y posteriormente ambas se agruparán en una sola con el color de la de la izquierda. Esto se repite hasta que todos los elementos del array se encuentran ordenados. Era imprescindible almacenar el color de cada elemento involucrado en cada paso, para poder restaurar los colores de forma correcta si retrocedemos en la ejecución.

Estados Merge Sort

Remarcar elementos - 0

Este estado se encarga de remarcar los elementos involucrados en esta iteración. Se crea cada vez que se llama a la función *merge* e involucra a los elementos comprendidos entre p y r. Para remarcarlos hemos optado por bajar su alfa a la mitad, por lo que los colores se transparentan, como se puede ver en la Figura 34

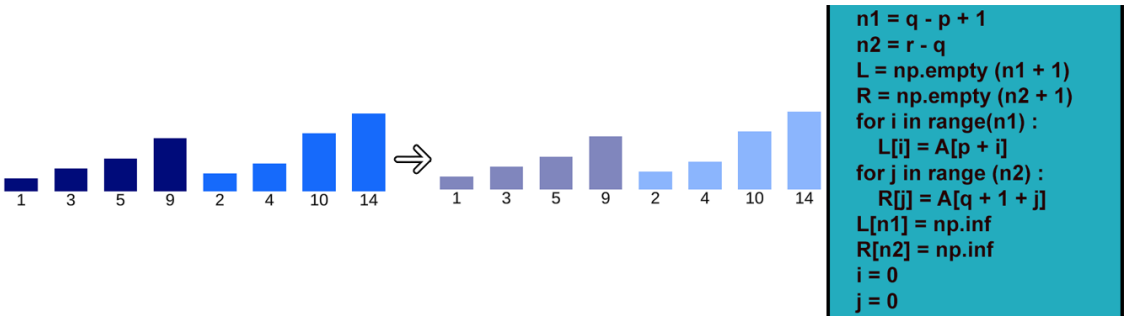

Figura 34: Estado remarcar elementos Merge Sort.

Mover elementos hacia abajo - 1

Este estado se encarga de mover hacia abajo los elementos, colocándolos en su posición ordenada dentro del rango de elementos que estemos trabajando. Como se puede ver en la Figura 35, pasamos de remarcar los elementos de la mitad izquierda, bajándoles su alfa, a ordenarlos con el color de la mitad izquierda en la parte inferior.

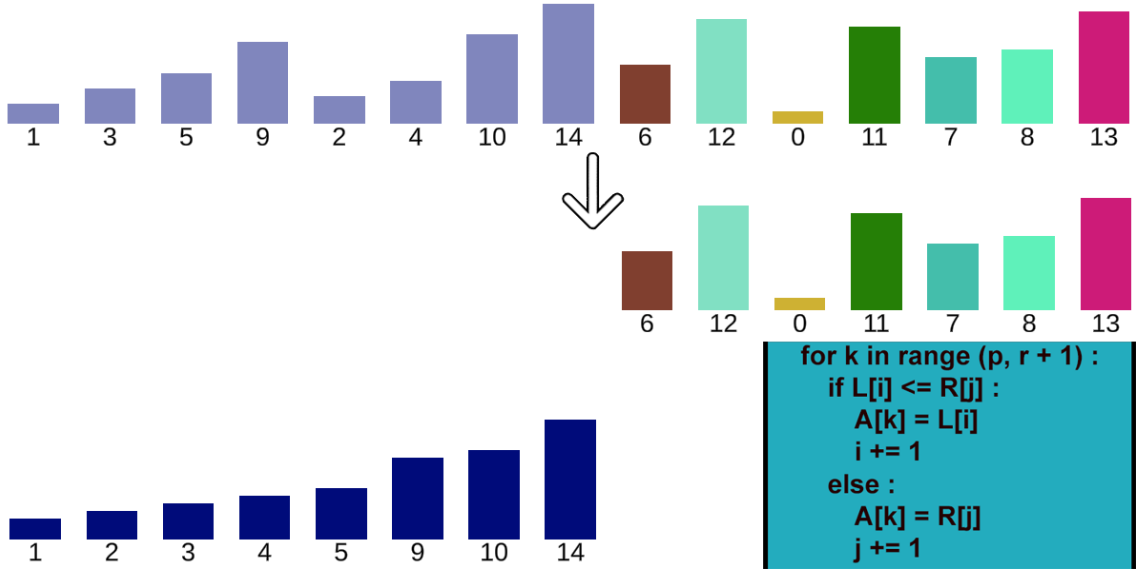

Figura 35: Estado mover elementos abajo Merge Sort.

Mover elementos hacia arriba - 2

Este estado simplemente se encarga de mover hacia arriba los elementos en sus posiciones correctas (ver Figura 36), gracias a los movimientos del estado 1 anterior.

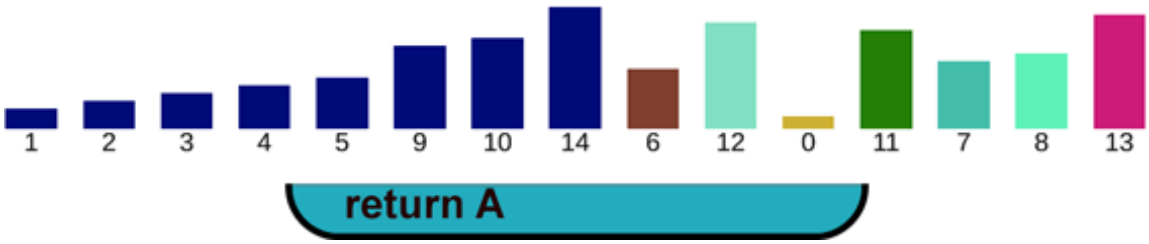

Figura 36: Estado mover elementos arriba Merge Sort.

Remarcar las llamadas en la función mergesort – 3

Este estado simplemente se encarga de remarcar, en el código de la función general mergesort, las llamadas que se realizan a la función merge y las distintas llamadas recursivas a la misma función. Se ha incluido este estado en los algoritmos recursivos para mejorar su comprensión al usuario. Podemos apreciar esto en la Figura 37.

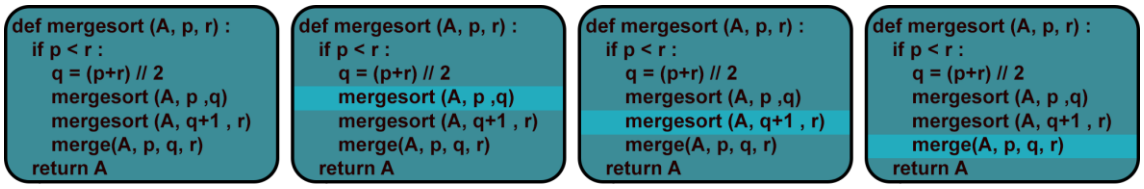

Figura 37: Estado remarcar las llamadas en la función mergesort.

Clase MergeSortState

Como en el resto de los algoritmos, el *Merge Sort* dispone de una clase propia que representa a un estado y contiene toda la información necesaria para reproducir de forma correcta la ejecución del algoritmo.

Propiedades como *StartIndex*, *EndIndex* nos ayudan a saber que subsecuencia estamos tratando. Por otro lado, en ciertos estados es necesario almacenar un color para poder realizar el paso atrás de forma correcta. Otras propiedades nos ayudarán a saber que números mover y a que posiciones y, por último, será necesario almacenar estados del array para realizar los cambios pertinentes y que la reproducción funcione correctamente.

4.9 Desarrollo algoritmo QuickSort

Quick Sort es el tercer algoritmo desarrollado para el proyecto. Su funcionamiento consiste en coger un elemento del array como pivote y colocar el resto de los elementos a un lado u otro del pivote en función de si son mayores o menores que él, organizando dos listas. Posteriormente se repite el proceso recursivamente para cada una de estas listas, siempre que tengan más de un elemento.

Este algoritmo comparte similitudes de implementación con los otros dos. Hemuna leyenda, ya que este algoritmo no es tan sencillo de entender visualmente como el de la burbuja o *merge sort*. Al igual que el resto de los elementos del proyecto, la leyenda se traduce automáticamente al idioma preferido por el usuario.

Estados Quick Sort

Elección y remarcación del pivote – 0

Este estado se encarga de pintar de color morado el elemento seleccionado como pivote, como se puede apreciar en la Figura 38.

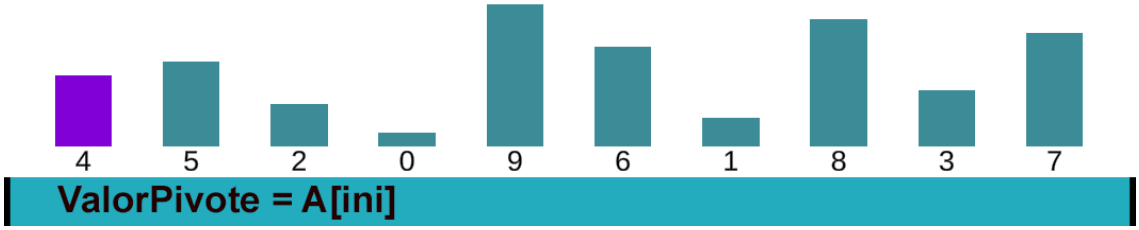

Figura 38: Estado elección pivote Quick Sort.

División elementos en 2 mitades: izq./menores y dcha./mayores que el pivote – 1 y 2

Este estado en realidad son 2. Cada uno de ellos se encarga de colorear el resto de los elementos de la secuencia de un color u otro, dependiendo de si son menores o mayores que el pivote, como se puede apreciar en la Figura 39.

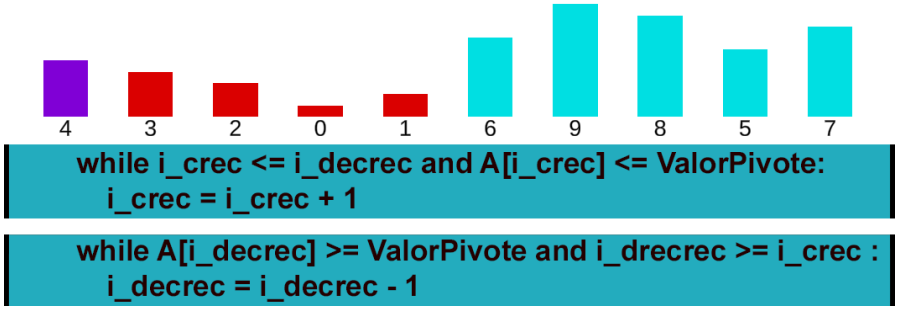

Figura 39: Estado división elementos en 2 mitades: izq./menores y dcha./mayores que el pivote Quick Sort.

Swap - 3

Este estado se encarga de realizar los intercambios de elementos que se producen durante la ejecución del algoritmo, como se puede ver en la Figura 40.

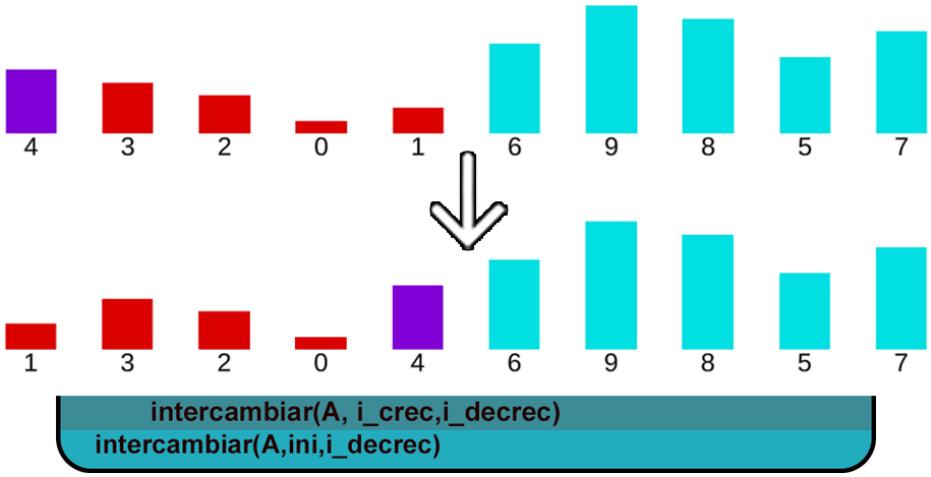

Figura 40: Estado swap Quick Sort.

Remarcar elemento ordenado – 4

Una vez ha realizado el ultimo intercambio de cada iteración, el elemento se encuentra ordenado y se marca de un color diferente que indica que se encuentra en su lugar correcto, como se puede apreciar en la Figura 41.

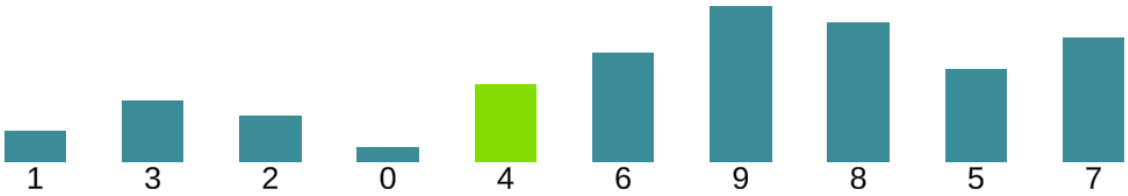

Figura 41: Estado remarcar elemento ordenado Quick Sort.

Remarcar las llamadas en la función quicksort – 5

Este estado simplemente se encarga de remarcar, en el código de la función general *Quicksort*, las llamadas que se realizan a la función pivote y las distintas llamadas recursivas a la misma función. Se ha incluido este estado en los algoritmos recursivos para mejorar su comprensión al usuario. Podemos apreciar esto en la Figura 42.

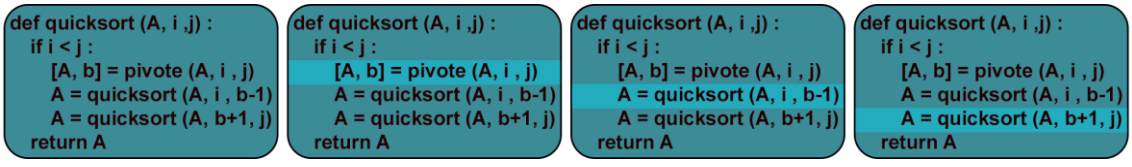

Figura 42: Estado remarcar las llamadas en la función Quicksort.

Clase Quick Sort State

Al igual que el resto de los algoritmos, el *Quick Sort* tiene una clase propia para representar sus estados y que sea posible la reproducción de su ejecución. Esta clase posee propiedades para garantizar el correcto funcionamiento de la reproducción. Estas propiedades tienen varios usos: remarcar elementos concretos, saber qué elementos intercambiar y distintas características de estos intercambios.

4.10 Desarrollo algoritmo de ordenación por selección e inserción

Para añadir más de variedad al proyecto y alternativas a los algoritmos de ordenación, se decidió incluir los algoritmos de ordenación por selección e inserción. Estos algoritmos son sencillos, menos eficientes que *Merge Sort* o *Quick Sort* y similarares en este aspecto al algoritmo de la burbuja. Comparten similitudes entre ambos, aunque poseen diferencias notables.

4.10.1 Desarrollo algoritmo de ordenación por selección (*selection sort*)

El desarrollo de este algoritmo fue el más sencillo de ambos. Su planteamiento consiste en buscar el mínimo elemento del array e intercambiarlo por el primer elemento, continuando así con el segundo, tercero, etc. Una vez que los elementos son intercambiados, los colocados a la izquierda no son considerados otra vez para ser el mínimo elemento. De esta forma, se va creando una cadena a la izquierda con todos los elementos ordenados mientras los de la derecha van siendo intercambiados a la mitad izquierda en orden.

Estados ordenación por selección (*selection sort*)

Remarcar el mínimo elemento – 0

Este estado se encarga de remarcar de color rojo el elemento que es considerado actualmente el mínimo. Esto puede cambiar y ser considerado otro elemento como el mínimo. Si esto ocurre, el elemento anterior vuelve a su color original y el nuevo se resalta de este color llamativo. Podemos apreciar, por ejemplo, en la Figura 43 que el mínimo elemento actual es el 0, el cual permanecerá como tal debido a ser realmente el menor del array. Por lo tanto, el 0 será movido a la primera posición en el próximo estado 3.

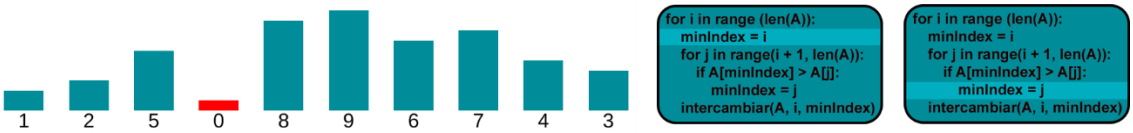

Figura 43: Estado remarcar mínimo elemento selection sort.

Remarcar el elemento a comparar – 1

Este estado se encarga de remarcar con azul el elemento que vamos a comparar con el mínimo elemento actual. Si este elemento es realmente menor, pasará a ser el nuevo mínimo actual. Como podemos apreciar en la Figura 44, el 2 es el mínimo elemento y se está comparando con el 1. Al ser menor, el 1 pasará a ser el mínimo actual.

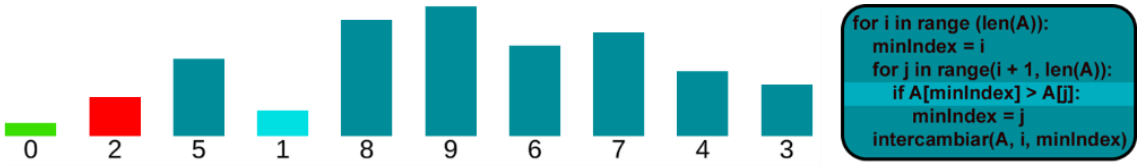

Figura 44: Estado remarcar elemento a comparar selection sort.

Remarcar el primer elemento sin ordenar – 2

Este estado, una vez conocido el mínimo elemento de la secuencia, se encarga de colorear del mismo color el primer elemento sin ordenar, para posteriormente intercambiarlo con el mínimo actual en el siguiente estado 3. Como podemos apreciar en la Figura 45, el 1 ha sido marcado como el mínimo elemento de la secuencia y se ha coloreado el primer elemento sin ordenar, que en este caso es el 2. Posteriormente serán intercambiados en el estado 3.

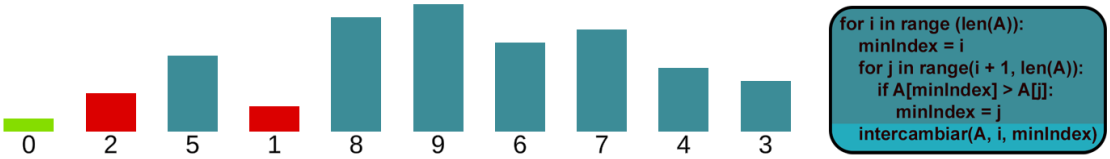

Figura 45: Estado remarcar primer elemento sin ordenar selection sort.

Intercambiar elementos – 3

Este estado se encarga de intercambiar el mínimo elemento con el primer elemento sin ordenar. Como podemos apreciar en la Figura 46, el mínimo elemento de la secuencia, el 1, ha sido intercambiado por el primer elemento sin ordenar, el 2. Posteriormente pasará a ser marcado como ordenado en el siguiente estado 4.

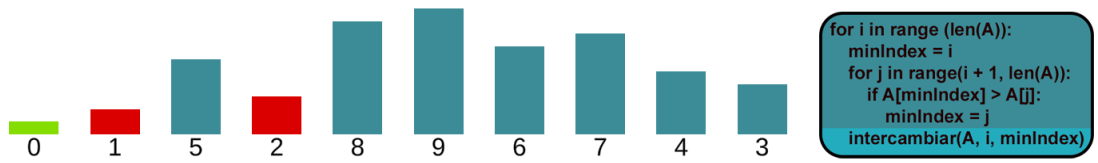

Figura 46: Estado intercambiar elementos selection sort.

Marcar elemento como ordenado – 4

Este estado se encarga de remarcar como ordenado, de color verde, el mínimo elemento intercambiado en el estado 3 anterior. Como podemos ver en la Figura 47, el mínimo elemento 1 ha sido intercambiado y marcado como ordenado.

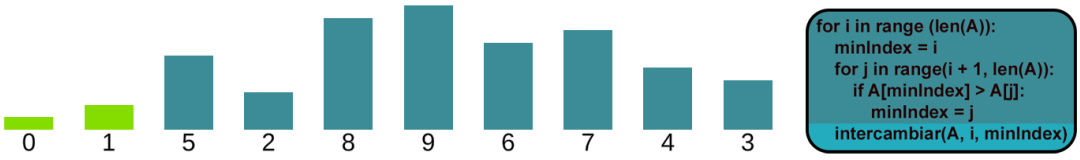

Figura 47:Estado remarcar como ordenado selection sort.

Clase SelectionSortState

La clase que representa a los estados del algoritmo de ordenación por selección es sencilla. Entre las propiedades necesarias para su correcto funcionamiento se encuentran: el índice del elemento a remarcar, el índice del elemento mínimo y un tipo que lo acompaña y, por último, los índices de los elementos a intercambiar.

4.10.2 Desarrollo algoritmo ordenación por inserción (*insertion sort*)

Como ya comentábamos anteriormente, este algoritmo posee similitudes con el de selección, aunque su desarrollo fue más laborioso. Esto es debido a que este algoritmo posee movimientos más complejos, similares a los del *merge sort*, con los que había que tener mayor precaución a la hora de respetar los cambios en el array. Para garantizar el correcto funcionamiento, ha sido necesario almacenar en ciertos estados referencias a los objetos de los elementos gráficos del array para realizar los cambios correctamente. Hablaremos de ello en profundidad en la sección de los estados.

El planteamiento de este algoritmo es seleccionar el primer elemento como ordenado y construir una cadena ordenada a partir de este. Es por ello por lo que, a continuación, se selecciona el primer elemento de la secuencia sin ordenar y se marca como clave. Posteriormente, esta clave se coloca en su lugar correcto dentro de la secuencia ordenada. Esto se repite hasta que no quedan elementos sin ordenar, cosa que se produce con el último elemento, que se encontrará ordenado siempre por ser el último.

Estados ordenación por inserción (*insertion sort*)

Marcar primer elemento como ordenado – 0

Este algoritmo es algo peculiar y funciona de forma similar al de selección, en cuanto a que va creando una cadena ordenada a la izquierda. Lo diferente es que siempre marca como ordenado el primer elemento y construye la secuencia ordenada a partir de este. Es por esto por lo que es necesario la existencia de un estado que solo se ejecuta una vez y remarcará este primer elemento como ordenado. Como podemos apreciar en la Figura 48, marcamos como ordenado el primer elemento, el 6 y construiremos nuestra secuencia ordenada a partir de este.

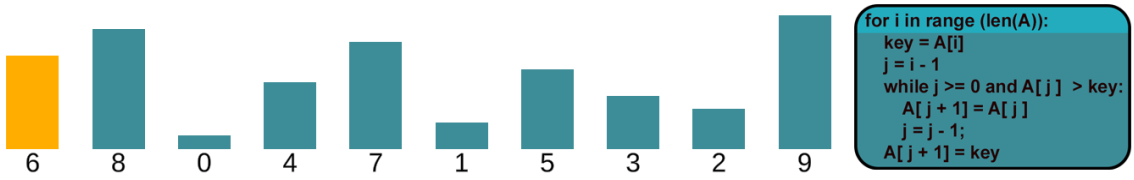

Figura 48: Estado remarcar primer elemento como ordenado insertion sort.

Remarcar la llave y moverla abajo – 1

Este estado se encarga de marcar el primer elemento de la secuencia sin ordenar como la llave y la mueve hacia abajo para prepararla para los movimientos que se producen en los estados siguientes. Como podemos apreciar en la Figura 49, el primer elemento sin ordenar es el 8, es marcado como la clave y se mueve abajo.

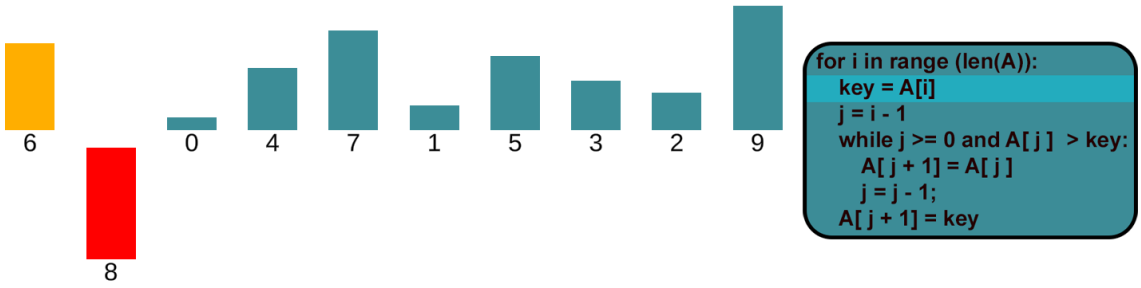

Figura 49: Estado remarcar la llave y moverla abajo insertion sort.

Intercambiar elementos – 2

Este estado se encarga de reproducir los intercambios que se producen en el array, adelantando los elementos más grandes que la clave. Como podemos ver en la Figura 50, la llave seleccionada es el 0. Al ser la llave menor que los 2 elementos de la secuencia ordenada, es necesario hacer 2 movimientos que corresponden cada uno a un estado 2 y colocarla en la posición del array que le corresponde. Estos cambios han de ser reflejados en la lista de los elementos gráficos del array y almacenar la referencia al objeto de la llave.

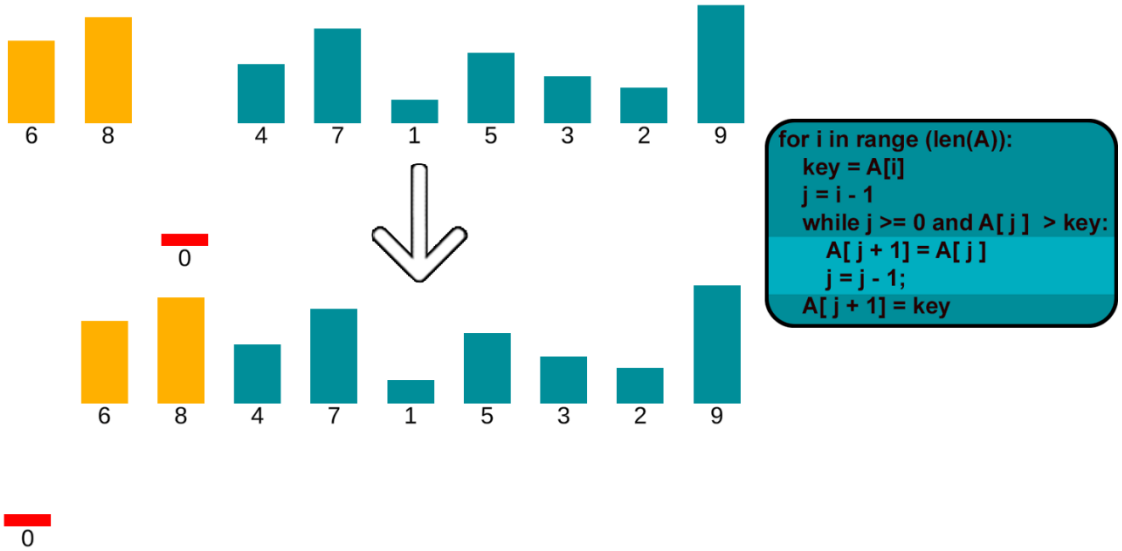

Figura 50: Estado intercambiar elementos insertion sort.

Mover la llave arriba y marcar el elemento como ordenado – 3

Por último, este estado se encarga simplemente de mover hacia arriba la llave, ya colocada en su posición correcta y marcar el elemento como ordenado. Como podemos ver en la Figura 51, nuestra llave, el 0, ha sido movida hacia arriba y marcada como ordenada con el color naranja.

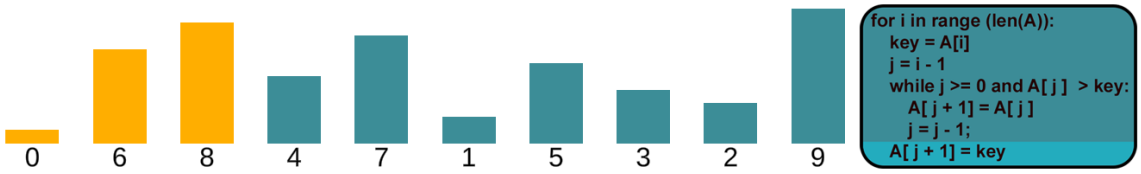

Figura 51: Estado mover la llave arriba y marcar elemento como ordenado insertion sort

Clase InsertionSortState

La clase que representa a los estados en este algoritmo es similar las demás. Posee propiedades cuyas funciones son: conocer el índice del elemento que debemos de remarcar, o hacia el que debemos de mover nuestro elemento. A su vez, será necesario almacenar ciertos elementos sustituidos, para revertir los cambios en el caso de que hagamos un paso atrás. Por último, será también necesario almacenar en ciertos estados el número llave para poder encontrar con este la referencia al objeto del elemento gráfico del array correspondiente.

4.11 Proyectos Git Hub y enlace de ejecución

Hemos empleado la herramienta Git Hub para el control de código fuente de la aplicación. Se han creado 2 proyectos. El primero es el del código del proyecto de Unity. El segundo, almacena las *builds* asociadas con cada uno de los *commits* del primero. Además, este segundo proyecto tiene añadida una Git Hub Page donde se ejecutará la última versión disponible del proyecto.

- Enlace del proyecto: https://github.com/jpguirado/V-Algo-Unity-Code
- Enlace de las *builds*: https://github.com/jpguirado/V-Algo-Unity-Builds
- Enlace para ejecutar la aplicación: https://jpguirado.github.io/V-Algo-Unity-Builds/

Capítulo 5 - Casos de uso

5.1 Ejecución del algoritmo de inserción directa

Una vez haya cargado el menú principal, como podemos ver en la Figura 52, hacemos clic en el algoritmo de inserción.

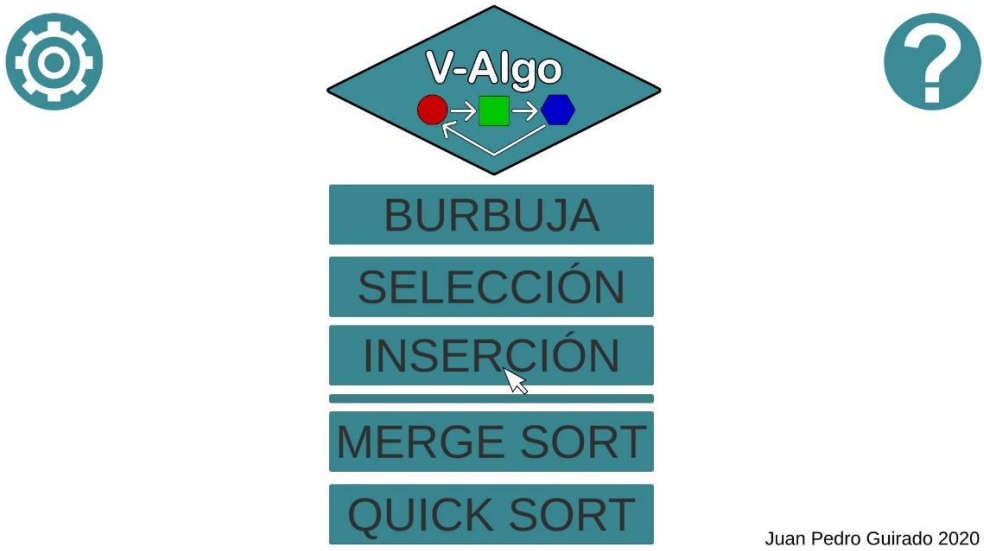

Figura 52: Menu principal caso de uso inserción.

Entraremos al menú de configuración de la ejecución y podremos elegir el número de elementos del array dentro de unos límites, como se puede apreciar en la Figura 53,

Figura 53: Menú configuración ejecución caso de uso merge sort.

Una vez dentro de la ejecución, la misma empezará a funcionar en modo automático y en velocidad media. Nosotros podremos pararla en cualquier momento y seguir paso por paso, retroceder o volver al inicio o al final de la ejecución. De la misma forma, podemos cambiar la velocidad de ejecución tanto del modo automático como de los movimientos de elementos en modo paso por paso. En cualquier momento podemos volver al menú principal, haciendo clic en el icono de la esquina superior izquierda. Podemos apreciar todo esto en la Figura 54, donde aparecen 2 pasos consecutivos de la ejecución.

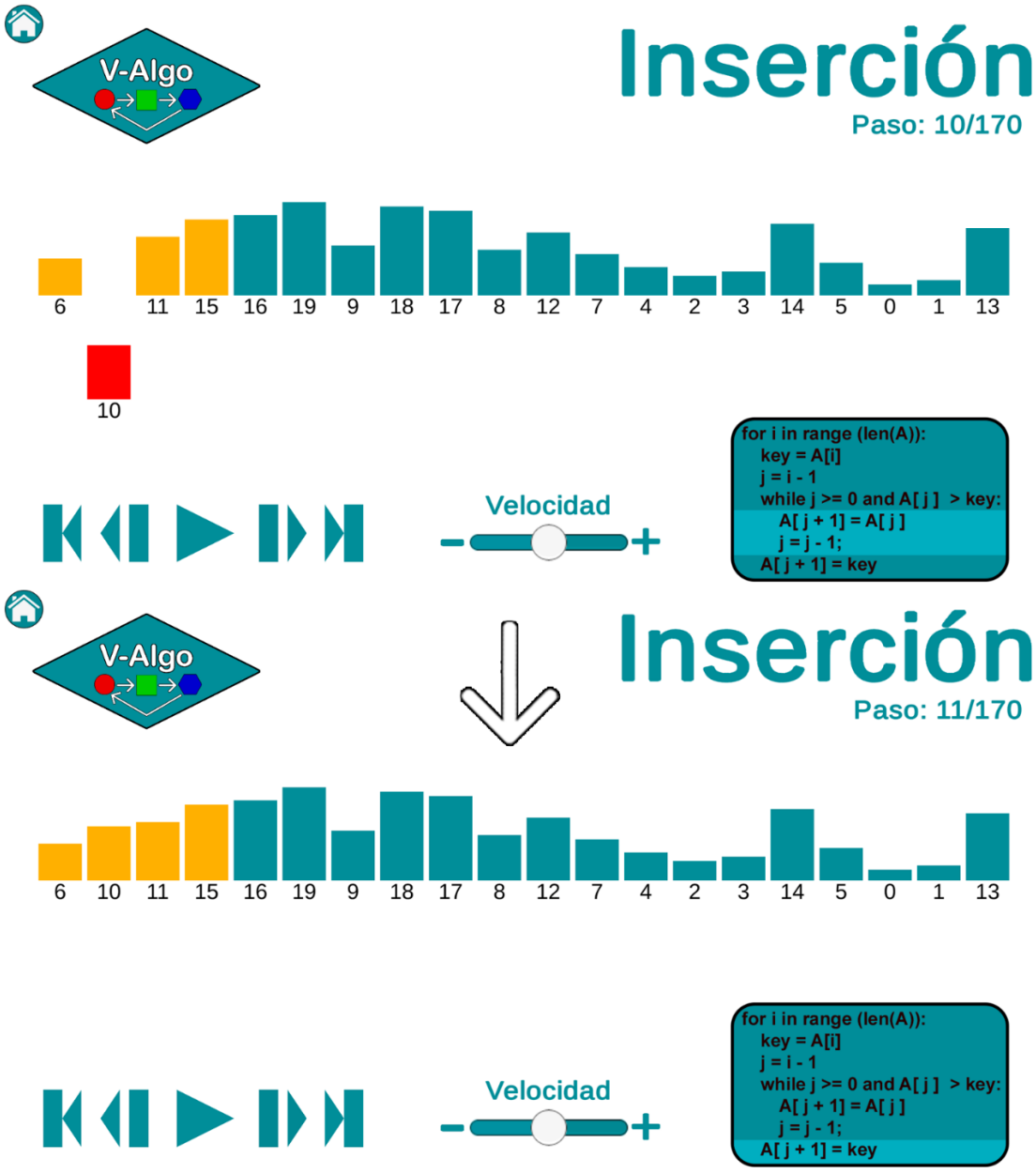

Figura 54: Ejecución algoritmo caso de uso inserción.

5.2 Ejecución del algoritmo de ordenación por mezcla

Una vez haya cargado el menú principal, como se puede observar en la Figura 55, hacemos clic en el algoritmo de *merge sort*.

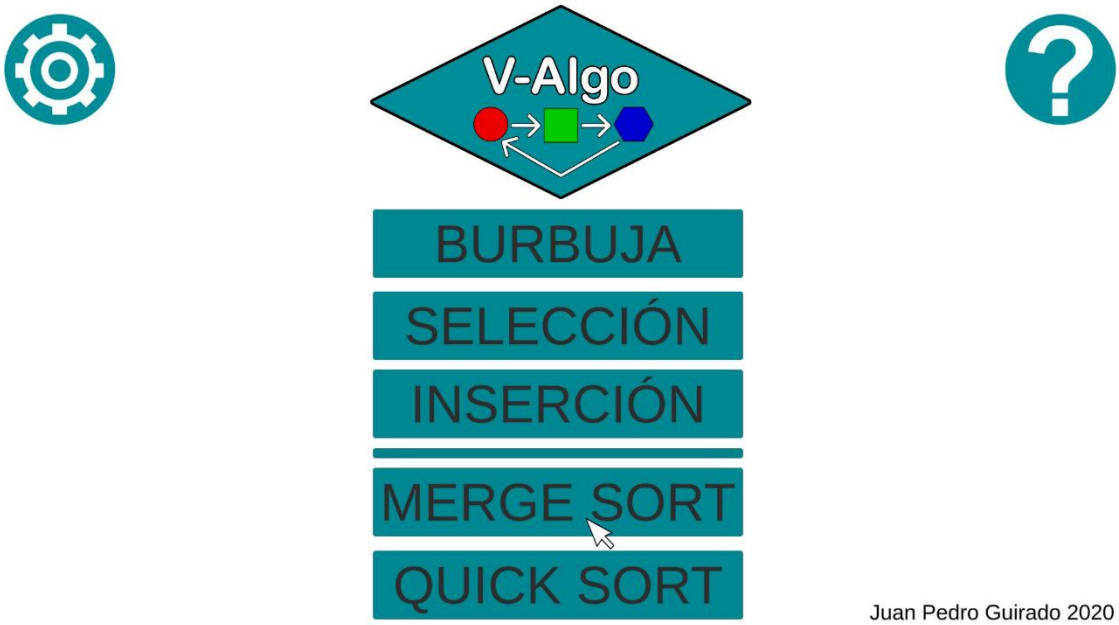

Figura 55: Menu principal caso de uso merge sort.

Entraremos al menú de configuración de la ejecución y podremos elegir el número de elementos del array dentro de unos límites. Este algoritmo al ser más complejo, su número máximo de elementos es 15, como podemos ver en la Figura 56.

Figura 56: Menú configuración ejecución caso de uso merge sort.

Una vez dentro de la ejecución, la misma empezará a funcionar en modo automático y en velocidad media. Nosotros podremos pararla en cualquier momento y seguir paso por paso, retroceder o volver al inicio o al final de la ejecución. De la misma forma podemos cambiar la velocidad de ejecución tanto del modo automático como de los movimientos de elementos en modo paso por paso. En cualquier momento podemos volver al menú principal haciendo clic en el icono de la esquina superior izquierda. Podemos apreciar todo esto en la Figura 57, donde aparecen 2 pasos consecutivos de la ejecución.

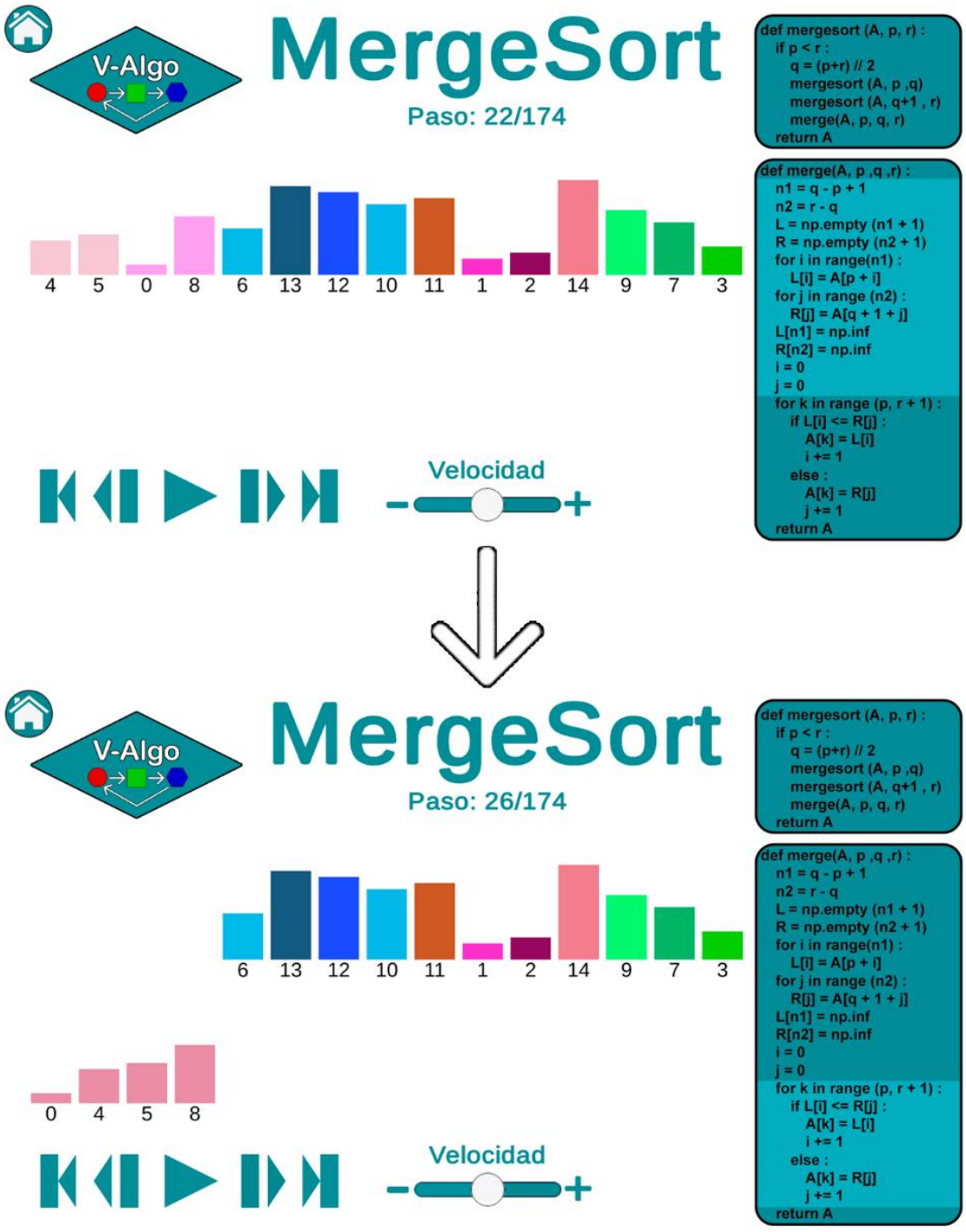

Figura 57: Ejecución algoritmo caso de uso merge sort.

Capítulo 6 - Conclusiones y trabajos futuros

6.1 Conclusiones

Una vez desarrollada la aplicación del visualizador de algoritmos podemos concluir que hemos alcanzado los objetivos con éxito. En primer lugar, consideramos que la aplicación resulta sencilla e intuitiva. Cuenta con una navegación simple y sin desbordar al usuario de información en ningún momento. En lo que se refiere a la ejecución de los algoritmos, el uso de diversos colores, junto con sus leyendas en los casos más complejos y los distintos movimientos, facilitan la comprensión del algoritmo por parte del usuario.

En segundo lugar, gracias a las facilidades de exportación de Unity a plataforma web y el gran potencial de crear Git Hub Pages [7] de forma sencilla, hemos podido hacer un despliegue de la aplicación rápido y efectivo. Esto permite que cualquier usuario con un navegador compatible pueda ejecutarla, sin necesidad de instalar ningún tipo de software. Otra de las ventajas de esto, es la posibilidad de subir actualizaciones de forma muy rápida, simple y sabiendo que se reflejará en todos los usuarios finales, ya que ellos no tendrán que hacer nada.

En tercer y último lugar, los dos puntos anteriores nos llevan a este, el de su uso académico. Consideramos que la elección de Python como lenguaje es muy acertada, debido a que es el que mayoritariamente se usa en el mundo de los algoritmos, ya sea para calculo, docencia y otros campos. Otro motivo es que su sintaxis es muy sencilla, cercana a un pseudo código y enfocada a que su lectura sea simple y eficaz, permitiendo su traducción a otros lenguajes sin problemas.

Otro punto para destacar es que, debido a la sencillez de su programación y como está organizada la misma, sumando el hecho de que es un proyecto público, permitirá a cualquier usuario añadir sus propios algoritmos, ya sean de ordenación o de cualquier otro tipo. De este aspecto hablaremos con más detalle en el siguiente apartado de trabajos futuros.

6.2 Trabajos futuros

De la forma que está organizado el proyecto, su carácter público y su plataforma de destino, consideremos que los trabajos futuros posibles están muy claros: el aporte de nuevos algoritmos y características a la herramienta.

Para este trabajo nos hemos centrado en los algoritmos de ordenación, debido a ser un problema muy común en el desarrollo de cualquier software y para aportar una variedad de soluciones al mismo problema con enfoques totalmente distintos. Es el momento pues, de que cualquier usuario pueda aportar sus visualizaciones de algoritmos, categorizándolos en problemas a resolver.

Para esta tarea, habría que reorganizar el menú para que pasase a presentar todas las categorías de algoritmos presentes en la herramienta, similar a lo que nos ofrece Visualgo[1]. Además, se podría aprovechar para dar un enfoque social a la plataforma, parecido a lo que hace Algomation [4], para que los usuarios puedan opinar sobre las visualizaciones, proponer otras alternativas al mismo algoritmo, etc.

Otra propuesta interesante sería la de permitir que el código fuese modificado por el usuario, de forma que le permitiese ver como estos cambios afectan a la ejecución del algoritmo. Algo similar a lo que nos ofrece Algorithm Visualizer [2], con un toque más de entorno de desarrollo.

Bibliografía

1. Hamlin S. 2011. Visualgo. Singapur. Disponible en https://visualgo.net
2. Park J. 2016. Algorithm Visualizer. Disponible en https://algorithm-visualizer.org/
3. Galles D. 2011. Data Structure Visualizations. Universidad de San Francisco. Disoponible en https://www.cs.usfca.edu/~galles/visualization/Algorithms.html
4. Meech D. 2014. Algomation. Disponible en http://www.algomation.com/
5. Bingmann Timo. 2013. 15 Sorting Algorithms in 6 Minutes. Disponible en https://www.youtube.com/watch?v=kPRA0W1kECg
6. Unity Technologies. 2005. Unity. Disponible en https://unity.com/es
7. Tom-Preston-Werner, Chris Wanstrath, P. J. Hyett, Scott Chacon, Microsoft. 2008. GitHub. Disponible en https://github.com
8. Pantrigo JJ. URJC. Temario asignatura Algoritmos para juegos 2018/2019