Variación del Knigt's Tour Mediante MVC

Alejandro Rodríguez, Rubén Palmer y Sergi Mayol

Resumen— Esta aplicación presenta una interfaz gráfica de usuario que le permite interactivamente solucinar el "Knight's tour" para un conjunto de piezas genéricas (Reina, Caballo, Torre, etc) podiendo escoger la pieza que guste y su posición inicial. Sin embargo, se ha modificado el algoritmo para que el recorrido pueda ser compartido por el conjunto piezas presentes en el tablero, por lo que se irán turando por el orden de inserción al tablero.

Vídeo - ver vídeo

1. Introducción

En este artículo explicaremos el funcionamiento y la implementación de nuestra práctica estructurando y dividiendo sus partes en los siguientes apartados:

Inicialmente, se citarán las librerías usadas en este proyecto. Para aquellas que se hayan desarrollado específicamente, se expondrá una breve guía de su funcionamiento y uso.

Seguidamente, se explicará cuál es la arquitectura, interfaces, funcionamiento e implementación de nuestro **M**odelo **V**ista **C**ontrolador (MVC) así como los métodos de mayor importancia. Para facilitar la legibilidad, se separará en cada uno de los grandes bloques de la estructura; en este caso Modelo, Vista, Controlador y Hub

Finalmente, se mostrará una breve guía de como usar nuestra aplicación y un ejemplo de esta.

2. Librerías

En esta sección se explicarán las librerías implementadas y usadas para llevar a cabo el desarrollo de las prácticas y permitir la reutilización de las mismas durante el transcurso de la asignatura.

El principal motivo que nos ha impulsado a implementar dichas librerías es la facilidad que proporcionan para centrarse únicamente en el desarrollo de la práctica en sí; Gracias a que durante el desarrollo de estas librerías se ha priorizado la facilidad de manejo, uso genérico y optimización para añadir el mínimo "overhead" al rendimiento del programa.

2.1. Better swing

Better swing [1] es una librería de derivada de java swing que permite un desarrollo más amigable y sencillo de interfaces gráficas, al estar inspirado en el desarrollo web (HTML y CSS), concretamente, en el framework de "Bootstrap". Se basa en "JfreeChart" para pintar las gráficas de líneas creando "wrappers" para facilitar su uso y manejabilidad.

2.1.1. Funcionamiento

El funcionamiento del paquete es muy sencillo, básicamente, la idea es crear una o varias ventanas con una configuración deseada y acto seguido ir creando y añadiendo secciones (componentes), donde posteriormente se pueden ir borrando y actualizando los componentes individual o grupalmente. Por ello, este paquete proporciona una serie de métodos para la ventana y las secciones.

¿Cómo funciona la ventana?

Para hacer que la ventana funcione es necesario crear una instancia del objeto `Window`, inicializar la configuración de la misma empleando el método `initConfig` recibiendo

por argumento la ruta donde se encuentra el archivo. En caso contrario se empleará la predeterminada. Finalmente, para que se visualice la ventana se realizará con el método `start` que visualizará una ventana con la configuración indicada anteriormente.

En el caso de querer añadir componentes como una barra de progreso, un botón o derivados, es tan sencillo como crear una sección y emplear el método `addSection`, el cual recibe por parámetro la sección a añadir, el nombre de la sección y la posición de este en la ventana.

¿Cómo funciona la sección?

Las secciones son instancias de la clase `Section` que permiten formar los diferentes componentes de la ventana.

El funcionamiento de una sección es muy sencilla, se trata de instanciar un objeto de la clase `Section` y llamar a algún método de esta clase, pasando los parámetros adecuados, y finalmente añadir la sección a la ventana.

Otros

Adicionalmente, se dispone de una clase llamada `DirectionAndPosition`, que constituye las posibles orientaciones y direcciones que una sección puede tener.

2.1.2. Implementación

Para el desarrollo de esta librería se ha centrado en la simplicidad hacia el usuario final y la eficiencia de código mediante funciones sencillas de emplear y optimizadas para asegurar un mayor rendimiento de la interfaz de usuario. Este paquete se divide en dos principales partes:

Window: Consiste en un conjunto de funciones para la creación y configuración de la ventana.

Section: Consiste en un conjunto de funciones base para la creación de secciones en la ventana.

La implementación de la `Window` consiste en diversas partes: gestión de teclas, configuración, creación y actualización de la ventana y creación de las secciones.

La gestión de las teclas se realiza mediante una clase llamada `KeyActionManager`, que implementa la interfaz `KeyListener`. Esta clase permite gestionar los eventos de teclado de la ventana y se utiliza para la depuración y el desarrollo del programa.

En cuanto, la configuración, creación y actualización de la ventana y la creación de las secciones, se han desarrollado una serie de métodos que envuelven a un conjunto de funciones propias de java swing. Por tanto, con este conjunto de métodos y funciones se obtiene la clase `Window`.

A continuación se explicarán los principales métodos de `Better swing`:

`initConfig` es un método que permite cargar la configuración de la vista y crea el marco de la vista, incluyendo apariencia, posición, tamaño, icono, color de fondo y manejo de eventos de teclado.

`start` permite la visualización de la ventana, haciendo que la ventana sea visible para el usuario.

`stop` actua como wrapper de `JFrame::dispose`.

`addSection` permite añadir una sección a un objeto `Window` indicándole la posición y dirección del panel mediante el conjunto de variables definidas en `DirectionAndPosition`.

`updateSection` permite actualizar una sección indicándole la posición y dirección del panel mediante el conjunto de variables definidas en `DirectionAndPosition`.

`deleteComponent` permite borrar un componente específico de la ventana que se le haya pasado por parámetro al método.

`repaintComponent` permite repintar un componente específico de la ventana que se le haya pasado por parámetro al método.

`repaintAllComponents` repinta todos los componentes de la ventana.

La implementación de la `Section` consiste en diversas partes, las cuales son las siguientes: Los métodos que permiten crear las secciones ya configuradas y las clases en las que se basan los métodos anteriores.

La propia librería contiene muchas más clases y métodos que si el lector desea explorar puede acceder a su documentación a través del código fuente proporcionado en la entrega de la práctica.

2.1.3. *Manual de uso*

En este apartado se describe como emplear la librería para su correcto funcionamiento.

Para emplear la librería es tan sencillo como crear una instancia de la clase `Window`, llamar al método `initConfig` indicando el fichero de configuración de la ventana, si se desea, en caso contrario se deberá pasar un `null` y se emplearán la configuración por defecto, y finalmente llamar a la función `start` cuando se desee inicializar la ventana.

Es importante inicializar la configuración antes de ejecutar la función `start`, ya que en caso contrario se producirá una excepción. A continuación se muestra un sencillo ejemplo:

```
1 Window view = new Window();
2 view.initConfig("config.json");
3 view.start();
```

Como se observa, la librería permite cargar la configuración e iniciar la ventana independientemente, permitiendo una mayor flexibilidad.

Además, en el caso de querer reiniciar la configuración, cambiar la visibilidad de la ventana o guardar el contenido de esta sin tener que volver a compilar el código, se puede realizar a través de unos atajos de teclado, que son los siguientes:

Q: Cerrar programa.
R: Reiniciar configuración.
V: Cambiar visibilidad.
G: Guardar contenido.

Por ejemplo, al cambiar la configuración y reiniciar la ventana se aplicarán los cambios automáticamente sin compilar de nuevo el código, es decir, se permite el conocido “Hot Reloading”.

En el caso de querer crear una sección y añadirla, se realizaría de la siguiente forma:

```
1  Window view = new Window();
2  Section section = new Section();
3  // Los datos pueden ser de longitudes irregulares
4  long[][] data = { ... };
5  Color chartColors[] = { Color.RED, Color.BLACK };
6  String chartColumnLabels[] = { "Linea 1",
7                                 "Linea 2" };
8  section.createLineChart(labels,
9                          data,
10                         chartColors,
11                         chartColumnLabels,
12                         "Ejemplo Lineas");
13 view.addSection(section,
14                 DirectionAndPosition.POSITION_TOP,
15                 "Chart");
```

En el ejemplo anterior se crearía una gráfica de líneas con dos líneas, una de color rojo y otra negra, con los nombres “Linea 1” y “Linea 2”, con el título “Ejemplos Lineas” y con los puntos del array data.

Finalmente, comentar que hay ilimitadas posibilidades de creación y personalización de componentes con los ya configurados y las posibilidades de crear libremente los propios componentes.

3. CONCEPTOS MVC

El Modelo Vista Controlador (MVC) es un patrón arquitectónico de software que divide una aplicación en tres elementos interconectados, separando la representación interna de la información, como se muestra al usuario y donde se hacen cómputos con esa información respectivamente.

Para esta práctica, se ha decidido modificar parcialmente este patrón al implementar la comunicación entre los módulos mediante un sistema de peticiones. Por ende, encontramos 4 módulos en nuestro MVC: Modelo, Vista, Controlador y Hub. Este último se encargará de gestionar toda la comunicación.

Esta decisión de modificación del MVC ha sido respaldada por la facilidad que obtenemos de escalar el patrón, permitiendo crear tantos módulos como queramos. Esto permitiría que, si en un futuro se quisiera añadir otro controlador en otro lenguaje de programación o encapsular controladores por funcionalidad, el único cambio a efectuar ocurriría en el hub.

A continuación se explicarán cada uno de los módulos y como han sido adaptados y empleados en la práctica.

3.1. Hub

El flujo de datos empleado en este patrón de diseño ha sido inspirado en como se comunican los diferentes elementos en un servidor. Es decir, cualquier módulo del MVC deberá realizar una petición al servidor (Hub) y este se encargará de solventar y dar servicio a dicha petición. Se puede ver con mayor claridad como funciona y su implementación en la siguiente figura:

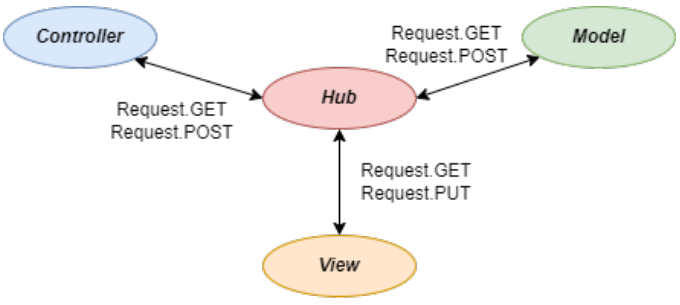

Figura 1. Patrón MVC usado en la práctica.

El Servidor/Hub se encarga de administrar los diferentes tipos de peticiones que llegan de los diferentes componentes. En la figura 1 se observan los diferentes tipos de peticiones posibles que se pueden realizar, en concreto: GET, POST y PUT. En un entorno web, GET sirve para obtener datos, POST para enviar y PUT para actualizar. En el programa los diferentes tipos de peticiones se han traducido en:

GET Realiza un retorno del dato solicitado.
POST Añade un el dato enviado.
PUT Realiza un set del dato enviado.

Para permitir esta implementación, se ha creado la interfaz `Notify` con el método `notifyRequest` el cual deberá estar implementado en todos los módulos del MVC, ya que será el responsable de gestionar las peticiones y asegurar que cada uno de ellos puede realizar sus tareas designadas de manera eficiente y precisa.

3.2. Modelo

El modelo es la representación de los datos que maneja el software. Contiene los mecanismos y la lógica necesaria para acceder a la información y para actualizar el estado del modelo.

Esta clase contiene diferentes tipos de estructuras de datos específicas a esta práctica, incluyendo, pero no solamente:

hub Permite la comunicación con el controlador y la vista
board Representa el tablero de ajedrez
boardWithMemory Array bidimensional de Piezas de ajedrez que representa el tablero de ajedrez con memoria, es decir, el tablero que mantien un registro de las jugadas anteriores.
iteration Representa el número de iteraciones que se han realizado en el algoritmo
elapsedTime Representa el tiempo que ha tardado el algoritmo en encontrar una solución

Finalmente, al ser un módulo de nuestro MVC, implementa la interfaz `Notify` y su método `notifyRequest` que le permite recibir notificaciones de los otros módulos del MVC.

3.3. Vista

La vista contiene los componentes para representar la interfaz del usuario (IU) del programa y las herramientas con las cuales el usuario puede interactuar con los datos de la aplicación. Adicionalmente, la vista se encarga de recibir e interpretar adecuadamente los datos obtenidos del modelo.

`loadContent` es el encargado de cargar el contenido inicial en la ventana. Para esta práctica, carga los siguientes elementos semánticos:

`createProgressBar`, función que crea y configura la barra de progreso de la ejecución del algoritmo, permitiendo dar feedback al usuario del estado del mismo.

`header` del programa mediante `headerSection`, generando las imágenes y la posible interación del usuario con las fichas, permitiendo así, elegir que ficha desea seleccionar para la ejecución del "Backtracking".

`main` del programa mediante `mainSection`, cargando el tablero con respecto a las variables por defecto encontradas en el modelo.

`sidebar` del programa mediante `sideBarSection`, generando las sección de estadísticas del programa.

`footer` del programa mediante `footerSection`, que ocupará la zona inferior de la interfaz, conteniendo, las acciones de iniciar y reinicar la ejecución del algoritmo.

Adicionalmente, la clase contiene un conjunto de métodos o funciones que facilitan la generación de los elementos en la interfaz de usuario. A continuación se explicarán brevemente los más importantes:

`getPieceListener` permite que el usuario seleccione una ficha del "header", además de crear una efecto de previsualización al pasar el cursor por encima de estas.

`Board`, esta clase permite generar el tablero de la interfaz a partir de un array bidimensional.

`BoxBoard`, esta clase permite generar cada casilla de la clase Board, además de permitir la interacción por individual de cada casilla con el usuario através de la función `getPieceListener`.

3.4. Controlador

El controlador en un MVC es el responsable de recibir y procesar la entrada del usuario y actualizar el modelo. En esta práctica, dicha responsabilidad es la de calcular el problema del "Knigt's tour" [5] para un conjunto indeterminado de piezas.

El algoritmo es una implementación recursiva con backtracking que toma los siguientes parámetros:

visitedTowns : Indica las posiciones que ya han sido visitadas.
pieces : Cola que representan las posiciones de las piezas que se usarán para el algoritmo y el orden de ejecución.
board : Representa el estado actual del tablero.
iteration : Lleva cuenta del número de iteraciones.

En primer lugar, se comprueba si se ha reiniciado la ejecución del algoritmo o si ya se han visitado todas las posiciones. A estas dos clausuras son las posibles condiciones de salida del algoritmo e impiden que este se ejecute de manera indefinida.

A continuación, eliminamos el primer elemento de la cola y se recorren en todos los posibles movimientos legales y no visitados de la pieza en esa posición.

Por cada movimiento, se actualiza esa posición como visitada, se modifica la cola de piezas añadiendo el nuevo movimiento, se ejecuta el movimiento en el tablero, se aumenta en uno la iteración y se crea una petición al hub para que el modelo pueda

obtener los resultados. Esto es posible debido a que, al ser un módulo del MVC modificado, implementa la interfaz `Notify` y el método `notifyRequest` que le permite comunicarse con los otros módulos.

Una vez salida de esa llamada recursiva, se revisa si el algoritmo se ha reiniciado para salir lo antes posible de este. En caso contrario, se deshacen los cambios previamente mencionados y se pasa al siguiente movimiento.

Una vez se han visitado todos los movimientos, al saber que no se ha encontrado una solución para esta llamada, se añade la posición eliminada al principio del algoritmo para mantener la estructura de los parámetros intacta.

La ejecución del algoritmo se realiza en un thread virtual, obteniendo una mejora en el tiempo de respuesta de la aplicación; permitiendo que otros eventos puedan tomar el thread principal, aprovechando los cores de la CPU y reduciendo el bloqueo de la aplicación; permitiendo que el thread principal sea menos probable de quedarse bloqueado debido a cálculos de larga duración.

Con respecto al tiempo computación asintótco, el algoritmo visita cada casilla del tablero como máximo una vez, por lo que obtendríamos $O(n^2)$.

Seguidamente, el algoritmo debe explorar, por cada casilla, el conjunto de movimientos de una pieza. Al tener una gran extensión de piezas y desconocer cuales se van a tomar, se tomará el peor caso posible en el que la pieza a escoger puede visitar todas las posiciones del tablero, lo que sería $O(n^2)$.

Para la obtención de los movimientos se toma una complejidad de $O(n)$ al crear un rango de movimientos en vez de recorrer toda la matriz, lo que quedaría como $O(n + n^2)$ por llamada recursiva, que se simplificaría a $O(n^2)$. Teniendo en cuenta todos los factores previamente comentados, podemos decir que la complejidad temporar del algoritmo es de $O\left(n^{2^{n^2}}\right)$ lo que es exponencial con respecto al número de casillas del tablero.

Adicionalmente, comentar que, debido a la naturaleza de los parámetros, no se llama directamente a este algoritmo, sino que primero pasa por una función que prepara todos los datos. De esta manera ocultamos los precálculos en otro método, creando una experiencia más limpia.

4. MANUAL DE USUARIO

Para el correcto uso de la aplicación, el conjunto de acciones que se pueden realizar en dicho programa serán definidas a continuación. También, la distribución de las secciones de la interfaz junto a su explicación y funcionalidad.

Cuando se ejecute el programa por primera vez en la pantalla se debe mostrar la siguiente interfaz de usuario:

Figura 2. Interfaz de usuario

4.1. Header

En el "header" de la aplicación podemos encontrar un conjunto de botones con imágenes que nos permiten seleccionar las piezas para poner sobre el tablero. Cuando se seleccione una pieza, ésta se verá reflejada en el cursor y se podrá añadir en cualquier punto del tablero clicando sobre una casilla.

Los botones del "header" para seleccionar las piezas, de izquierda a derecha, son:

Rey	Álfil
Reina	Unicornio
Torre	Dragon
Caballo	Castillo

4.2. Main

En el principal bloque de la aplicación se encuentra el tablero donde se ejecutará el algoritmo de backtracking. En esta sección, se representa el tamaño del tablero, las piezas que hay en el tablero y las casillas.

Añadir que en la parte derecha del tablero se podrán consultar las estadísticas, donde se da información acerca del tamaño del tablero, piezas que hay sobre el tablero, tiempo transcurrido y progreso del algoritmo.

4.3. Sidebar

En el "Sidebar" de la apliación se encuentran las estadísticas del de la ejecución. En estas estadísticas se indica el tamaño del tablero, el número de piezas que hay sobre el tablero, el tiempo transcurrido en encontrar la solución y el progreso del algoritmo se verá reflejado en la barra de progreso.

4.4. Footer

En el "footer" de la aplicación se pueden encontrar un conjunto de "gadgets" que nos permiten controlar la ejecución del programa y visualización. A continuación se explicará cada uno de ellos de izquierda a derecha:

Primeramente, el botón definido por la etiqueta "Iniciar" permite iniciar el algoritmo con la configuración que el usuario haya elegido, es decir, se iniciará con el tamaño del tablero seleccionado y con N piezas sobre el tablero que el usuario habrá indicado donde comienza cada una. Si se pulsa el botón "Iniciar" sin que haya ninguna pieza sobre el tablero, saltará un Error diciendo que no hay piezas suficientes piezas en el tablero.

Por otra parte, el botón definido por la etiqueta "Reiniciar" permite reiniciar el algoritmo y el tablero y dejarlo como al principio de la ejecución del programa. El usuario deberá introducir otra configuración de piezas y de tamaño de tablero para volverlo a ejecutar.

Por último, a la derecha de éste hay un objeto JSpinner para seleccionar el "Tamaño del tablero", cada vez que se cambie el tamaño del tablero, el cambio se verá reflejado sobre el tablero representado por pantalla. Como mínimo, el tablero tiene que ser de 2x2 y como máximo 32x32. Queda claro que el algoritmo tardará menos en ejecutarse en un tablero de 2x2 que en un tablero de 32x32, ya que se tienen que recorrer muchos menos puntos.

4.5. Ejemplo ejecución

Al iniciar la aplicación se mostrará una interfaz como la expuesta en la imagen 2. Para poder iniciar la ejecución del algoritmo, es necesario que se coloquen 1 o más piezas en el tablero de diferentes tipos, como se ha mencionado anteriormente en la sección 4.1. Una vez ya seleccionada las N fichas, se debe pulsar el botón de iniciar, provacando que se inicie el algoritmo y viendo a tiempo real la ejecución del "backtracking", junto a los datos de la ejecución a la derecha.

A continuación, la siguiente imagen, muestra un ejemplo de la interfaz en mitad de la ejecución:

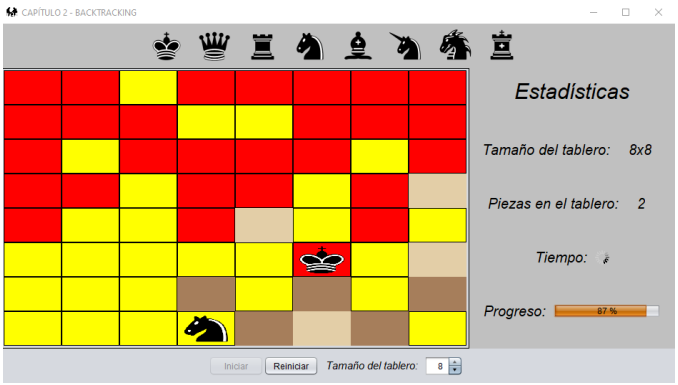

Figura 3. Interfaz de usuario

A partir de aqui, el usuario puede reiniciar la ejecución para ejecutar nuevamente el algoritmo.

5. Conclusión

Teniendo en cuenta el desarrollo expuesto, la aplicación permite generar un punto de referencia de un conjunto de algoritmos de diferente complejidad asintótica mediante una representación gráfica e interactiva de la media de los tiempos de ejecución a través de unos parámetros de entrada definidos por el usuario. Adicionalmente, se ha implementado mediante una versión modificada del patrón de diseño de software **M**odelo **V**ista **C**ontrolador (MVC) añadiendo un cuatro módulos que permite la comunicación entre los diferentes elementos de este mediante peticiones. Esta modificación ha sido fuertemente inspirada en el diseño de un servidor, lo que permite gran flexibilidad y escalabilidad de desarrollo, al poder interceptar y gestionar las peticiones a gusto del desarrollador.

Además, se ha hecho énfasis en el uso de las librerías creadas por los miembros del grupo para facilitar el trabajo y la reutilización de código en futuras prácticas. Reiterar que el principal motivo que nos ha impulsado a implementar dichas librerías es la facilidad que proporcionan para centrarse únicamente en el desarrollo de la práctica en sí; Gracias a que durante el desarrollo de estas librerías se ha priorizado la facilidad de manejo, uso genérico y optimización para añadir el mínimo "overhead" al rendimiento del programa.

Concluir que, con respecto al ámbito académico, este proyecto nos ha permitido consolidar el concepto de patrón de diseño MVC gracias a un primer proceso de investigación y discusión del diseño a implementar entre los integrantes del grupo.

6. Distribución del trabajo realizado

El trabajo realizado por cada miembro de la práctica ha sido el siguiente:

- Alejandro Rodríguez:
  - Documentación.
  - Principal desarrollador del "frontend" de la aplicación.
  - Desarrollador de funcionalidades extra de la IU.
  - Desarrollador del "backend" de la aplicación
  - Desarrollador general de la aplicación.
  - Desarrollador del patrón MVC de la aplicación.
- Rubén Palmer:
  - Documentación
  - Principal desarrollador del "backend" de la aplicación
  - Desarrollador general de la aplicación
  - Desarrollador del "frontend"
  - Diseñador de la implementación del patrón MVC de la aplicación
  - Desarrollador de la plantilla base del proyecto
  - Desarrollador de la plantilla de la documentación
- Sergi Mayol:
  - Documentación
  - Desarrollador de la librería `Better Swing`
  - Principal desarrollador del "frontend" de la aplicación
  - Desarrollador general de la aplicación
  - Desarrollador del "backend" de la aplicación
  - Diseñador de la implementación del patrón MVC de la aplicación
  - Desarrollador de la plantilla base del proyecto

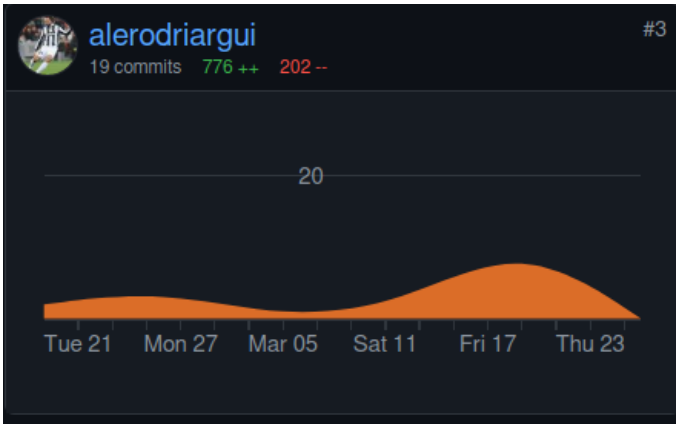

Figura 4. Contribución en el proyecto de Alejandro Rodríguez

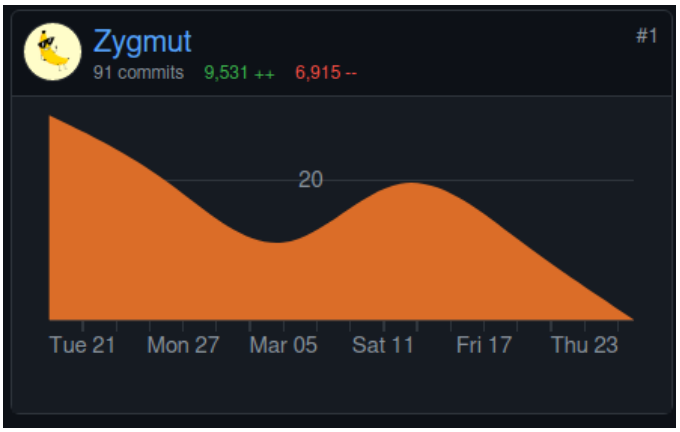

Figura 5. Contribución en el proyecto de Rubén Palmer

Figura 6. Contribución en el proyecto de Sergi Mayol

Reconocimientos

- Se agradece la colaboración del Dr. Miquel Mascaró Portells en la resolución de dudas y supervisión del proyecto.

Referencias

[1] Better Swing, *An easy way to develop java GUI apps*, V0.0.2. Ver documentación completa aquí.

[2] Graphical engine, *How to develop a graphical engine from scratch*. Ver enlace aquí.

[3] Chesscraft. Obtenido 25 Marzo, 2023, desde web

[4] Chess Programming Wiki. Move Generation. Obtenido 25 Marzo, 2023, desde web

[5] A. J. Schwenk, "The knight's tour,"*Mathematics Magazine*, vol. 67, no. 1, pp. 11–22, 1994.

[6] D. B. West, *Introduction to Graph Theory*. Prentice hall, 2001.

[7] R. M. Haralick, "The environment and its effects on systems performance,ïn *Proceedings of the fifth international joint conference on Artificial intelligence-Volume 1*, 1981, pp. 569–573.

[8] R. E. Korf, "Finding optimal solutions to the twenty-four puzzle,"*Artificial Intelligence*, vol. 103, no. 1-2, pp. 1–24, 1998.