Number Match

Asignatura: **Introducción a la programación**

Curso: **2022/2023**

Estudiante: . José Manuel de Torres Domínguez .

DNI: . [redacted] .

Estudiante: . Manuel Alonso González .

DNI: . [redacted] .

Grupo de prácticas: 4

Profesor/a de prácticas: Cristina Vicente

Convocatoria: Ordinaria

ÍNDICE

1. Introducción

En este documento se recoge toda la información relacionada con la planificación, implementación y compilación del proyecto Number Match, así como sus posteriores modificaciones en base a las ampliaciones sugeridas en la rúbrica de calificación del proyecto.

Este documento se divide en varias secciones en las que se detallaran de forma precisa, añadiendo gráficos e imágenes siempre que se considere necesario, la planificación, el código, y otros contenidos relacionados con la implementación de este proyecto.

El juego tiene un formato de tablero. El tablero al comienzo de la partida tiene tantas filas ocupadas con números como se le haya estipulado al comienzo. Para lograr la victoria se deberá vaciar el tablero. El juego consiste en hacer parejas de números, de tal forma que cuando dos números hacen pareja, éstos sean borrados. A su vez, si eliminamos un número y queda la fila borrada, se eliminará esa fila y todas las inferiores ascenderán una o dos posiciones, depende del número de filas que se borren. Para que dos números formen pareja, éstos deben ser iguales o sumar diez. También es importante la posición a la hora de hacer las parejas, ya que solo la harán aquellos números que estén en posición horizontal, vertical, diagonal o uno esté en la última celda de una línea y el otro en la primera de la siguiente fila. Hay una mecánica llamada “réplica” que nos sirve para el momento en el que nos quedemos sin parejas disponibles. Al utilizar la “réplica” lo que hará el juego es copiar a partir de la primera celda vacía todos los números que quedan en el tablero, uno seguido del otro. De esta manera, se podrán crear nuevas parejas. El juego cuenta con un número limitado de “réplicas”, que también será definido por el jugador antes de empezar a jugar. El juego puede acabar de diversas formas: pulsando la tecla “Escape”, que conllevará la salida del juego; si te quedas sin “réplicas”, que supondrá que has perdido; y si intentamos “replicar” pero no quedan suficientes celdas útiles para contener todos los números que necesitan ser copiados.

En cuanto a los requisitos mínimos que se piden para el correcto funcionamiento del programa, hemos hecho la implementación correcta y que solventa con éxito los problemas encontrados y consigue que el programa funcione correctamente y con fluidez.

El IDE utilizado para llevar a cabo este proyecto ha sido Eclipse CDT para C++ (2019). Además, en el ámbito de la programación, se ha utilizado la extensión de Eclipse multiplataforma, *CodeTogether*, una herramienta de colaboración en tiempo real, que se detallará con más profundidad al final del documento.

2. Análisis y diseño

2.1. Análisis

El punto de inicio del proyecto da comienzo en el análisis de la versión de prueba suministrada por el profesorado de IP, así como la extensa y detenida lectura, comprendiendo los distintos objetivos y ampliaciones propuestos.

El objetivo básico obligatorio consta de la réplica del juego Number Match, con numerosas adaptaciones en diferentes sistemas.

Para poder plantear el entorno gráfico, así como el funcionamiento del juego con su conjunto de reglas, es necesario la implementación de un conjunto de herramientas básicas que nos ayude en la gestión y manipulación de los datos del juego, así como la comunicación entre la librería del entorno gráfico provisto y el programa encargado de la ejecución y gestión de la lógica del juego.

Este conjunto de herramientas está compuesto por 3 componentes principales:

- TAD Celda: Encargado de gestionar individualmente la información, escritura y lectura de los valores de las celdas del tablero.

- TAD Tablero: Encargado de gestionar la matriz de celdas, así como las propiedades de está y los métodos para su escritura y lectura de datos.

- TAD Juego: Encargado de gestionar la lógica y las reglas del juego, así como recibir y responder a los comandos del usuario e interconectar el entorno gráfico con la información de la sesión actual.

Este conjunto de herramientas nos permite poder llevar a cabo las diferentes funciones del juego de una manera rápida, sencilla, ordenada y efectiva, sin tener que sobrecargar de código las diferentes partes del programa y así, minimizar el código que se necesita ejecutar en el hilo de procesamiento del programa principal.

Además, también se usa una librería externa de terceros “Allegro” que, junto a una matriz de métodos personalizada provista por el profesorado de IP, permite la representación del juego mediante un entorno gráfico, a modo de tablero.

2.2. Diseño

Para poder desarrollar eficazmente un programa complejo, como lo es un juego, es necesario tener preparado un plan de desarrollo y una guía para diseñar el código de manera estructurada.

El diseño del código del juego está estructurado como una pirámide: Los TADs
más básicos y sencillos fundarán las bases para los siguientes TADs, que realizan operaciones más complejas y requieren (por lo general) de más líneas de código.

Como hemos explicado anteriormente: *"El juego tiene un formato de tablero. El tablero al comienzo de la partida tiene tantas filas ocupadas con números como se le haya estipulado al comienzo."*

Como todo el juego gira alrededor del uso de un tablero, será crucial el programar una serie de métodos que nos permitan poder modificar las celdas y sus propiedades. Este TAD recibirá el nombre de TAD Celda. El siguiente TAD, que se funda en las bases del anterior, es el que realizará una lista de operaciones más complejas para ser utilizados en la creación de la lógica del juego, utilizando los métodos encontrados en el TAD Celda. Este TAD recibe el nombre de TAD Tablero. Finalmente, el último TAD se basa en el anterior TAD, que a su vez se funda en las bases del TAD Celda. Este TAD debe ejecutar la lógica y las reglas del juego. Por tanto, este TAD recibe el nombre de TAD Juego.

Una vez se han definido todos los TADs, simplemente queda la ejecución del juego (que reside en el TAD Juego) mediante su invocación en el programa principal.

2.3. Diagrama modular

En cuanto al método de programación utilizado en este proyecto, hemos utilizado un método basado en la programación modular, ya que consideramos este método como una forma bastante clara de estructurar el código y ayuda a la legibilidad del código, además de facilitarnos el trabajo de escribir código.

2.4. TAD Celda

Este TAD es el utilizado para modificar las celdas de nuestro tablero con operaciones como borrar un número o poner un número determinado en una celda concreta. Además, servirán de elementos para la matriz que guarda la información del tablero.

> TAD Celda: crearCelda, crearCeldaVacia, borrarNumeroCelda, obtenerNumeroCelda, esVaciaCelda, esBorradaCelda, sonParejaC.

Operaciones:

crearCelda (c: celda, n: entero)
modifica c.
efecto Inicializa una celda y le asigna el número n.

crearCeldaVacia (c: celda)
modifica c.
efecto Inicializa la celda dada con los valores vacíos.

borrarNumeroCelda (c: celda)
modifica c.
efecto Marca la celda como borrada.

obtenerNumeroCelda (c: celda)
salida entero.
efecto Devuelve el número contenido en la celda.

esVaciaCelda (c: celda)

salida bool.
efecto Devuelve 'true' en caso de que la celda está vacía, 'false' en caso contrario.

esBorradaCelda (c: celda)

salida bool.
efecto Devuelve 'true' en caso de que la celda está vacía, 'false' en caso contrario.

sonParejaC (c: celda, d: celda)

salida bool.
efecto Devuelve 'true' en caso de que los números sean iguales, 'false' en caso contrario.

2.5. TAD Tablero

Este TAD es el contiene la información del tablero, tanto las celdas como otros valores tales como el número de celdas ocupadas. Es este TAD sobre el que se realizan todas las operaciones a medida que se dicten en el TAD Juego conforme avanza la partida.

TAD Tablero: crearTablero, vaciarCelda, borrarCelda, obtenerNum, estaVacia, estaBorrada, borrarFila, mostrarTablero, obtenerFilCol, obtenerFilColMax, devolverCoordUltimaCeldaOcupada, ponerNumCelda, sonPareja, esFilaBorrada, replicar.

Operaciones fundamentales:

crearTablero (t: tablero, filas: entero, columnas: entero, iniciales: entero)

modifica t.
efecto Inicializa el tablero dado, con dimensiones 'filas' x 'columnas', rellenando con números aleatorios (valores entre 1 y 9) la cantidad de filas indicada por 'iniciales'.

Si filas es 0, se utilizarán las dimensiones por defecto del entorno.
Si columnas es 0, se utilizarán las dimensiones por defecto del entorno.
Si iniciales es 0, se devolverá un tablero vacío.

vaciarCelda (t: tablero, fil: entero, col: entero)

modifica t.
efecto Devuelve el tablero con la celda seleccionada vacía.

borrarCelda (t: tablero, fil: entero, col: entero)

modifica t.
efecto Devuelve el tablero con la celda seleccionada borrada.

obtenerNum (t: tablero, fil: entero, col: entero)
salida entero.
efecto Devuelve el número de la celda seleccionada.

estaVacia (t: tablero, fil: entero, col: entero)
salida bool.
efecto Devuelve 'true' si la celda seleccionada está vacía, 'false' en caso contrario.

estaBorrada (t: tablero, fil: entero, col: entero)
salida bool.
efecto Devuelve 'true' si la celda seleccionada esta borrada, 'false' en caso contrario.

borrarFila (t: tablero, fil: entero)
modifica t.
efecto Borra la fila 'fil' del tablero y mueve las que tenga por debajo una posición hacia arriba.

obtenerFilCol (t: tablero, filaS: entero, colS: entero)
modifica filaS, colS.
efecto Devuelve el número de filas y columnas útiles.

obtenerFilColMax (t: tablero, filaM: entero, colM: entero)
modifica filaM, colM.
efecto Devuelve el número de filas y columnas máximas del tablero.

devolverCoordUltimaCeldaOcupada (t: tablero, coordFilaM: entero, coordCol: entero)
modifica coordFila, coordCol.
efecto Devuelve las coordenadas de la última celda ocupada en el tablero.

ponerNumCelda (t: tablero, fil: entero, col: entero, num: entero)
modifica t.
efecto Ajusta un número concreto a una celda de una fila y una columna especificadas.

sonPareja (t: tablero, fil1: entero, col1: entero, fil2: entero, col2: entero)
salida bool.
efecto Devuelve 'true' si las celdas especificadas forman una pareja. En caso contrario, devuelve 'false'.

esFilaBorrada (t: tablero, fil: entero)
salida bool.
efecto Devuelve 'true' si todas las celdas de la fila especificada están borradas. De lo contrario, devuelve 'false'.

replicar (t: tablero)

salida t.
efecto Copia en serie a partir de la última celda ocupada todas las celdas ocupadas restantes en el tablero (incluida ella misma).

Devuelve 'true' si se ha completado la operación correctamente, y 'false' en caso contrario.

Operaciones adicionales:

mostrarTablero (t: tablero)

efecto Muestra el tablero 't' en el terminal. Si una celda está borrada, mostrará el número entre paréntesis (), y si está vacía, mostrará una 'V'.

2.6. TAD Juego

Este es el TAD encargado de hacer que el entorno y el tablero vayan a la vez. A su vez, es el encargado de que el juego se inicie y cierre correctamente, así como el desarrollo de la partida.

TAD Juego: iniciarPartida, tickPartida y terminarPartida.

Operaciones:

iniciarPartida (p: partida)

salida bool.
modifica p.
efecto Carga el entorno e inicializa todas las variables necesarias de los diferentes TADs.

tickPartida (p: partida)

salida entero.
modifica p.
efecto Se encarga de ejecutar la partida y hacer los cambios necesarios en el tablero.

terminarPartida (exit: entero)

efecto Finaliza el entorno y la partida. La manera en la cual se ha producido la salida del programa viene dada por exit.

2.7. Programa principal

En nuestro programa principal, lo primero que hacemos es declarar dos variables, *exit* y *p*. La variable *exit* será el entero con el que se define de qué modo se acabó el juego, como la victoria o porque le dan a la tecla “Escape”.

Primero se ejecuta el iniciarPartida del TAD Juego que, en caso de devolver *true* significará que se logró iniciar correctamente el entorno y entonces entrará en el bloque condicional. Dentro del bloque condicional se encuentra el módulo tickPartida, que se encarga de ejecutar la partida.

Una vez que el tickPartida termine, devolverá un entero que se le asignará al parámetro *exit*. Éste entrará como parámetro de entrada al último módulo, que es terminarPartida. Éste se encargará de mostrar el mensaje final acorde al parámetro *exit* y cerrará el entorno correctamente, a la vez que terminará el problema.

2.8. Ampliaciones

2.8.1. Ampliación 1: Parejas separadas

En esta ampliación modificamos el módulo sonPareja del TAD Tablero, de tal manera que llevase a cabo la comprobación de las nuevas reglas que determinan los nuevos tipos de parejas.

Para esta modificación, decidimos dejar el módulo original y, en caso de que no formasen pareja en el conjunto de reglas base, entonces entraría en un nuevo código que comprueba, no solo la posición sino también si las celdas que se encuentran entre ambas celdas se encuentran borradas.

Para ello, con una serie de condiciones anidadas se especifica la posición relativa de ambas celdas y, conforme su posición, se comprueban unas celdas u otras.

2.8.2. Ampliación 2: Ayuda

En esta modificación, incluimos el código que se ejecutará cuando se presione la tecla “F2”. Este código se encuentra dentro del módulo tickPartida, dentro del bucle y dentro del condicional *switch*, en el caso de “F2”.

El código consiste en cuatro bucles anidados, cuya función es comprobar cada número con el resto del tablero y, con el primero con el que forme pareja, la marca de amarillo para marcar que son pareja. Si no encuentra ninguna pareja, se muestra un mensaje en la parte inferior del entorno. También hay un límite de ayudas que puede solicitar el jugador, lo cual está implementado con éxito.

2.8.3. **Ampl*iación 3: Juego automático***

En esta ampliación, modificamos el módulo de tickPartida en el TAD Juego. De tal manera que el módulo no necesitase leer teclas para actuar.

Tomamos la decisión de coger el código de la ampliación de la ayuda y ponerla al principio del bucle. Ésto servía para que se escogiesen las coordenadas de dos celdas que formaban parejas y a continuación, éstas se guardaban en dos parámetros.

Éstos se introducían en el código que había anteriormente en el condicional *switch*, en el caso de la tecla “Enter”. Si no había conseguido ganar, entonces empezaba de nuevo.

Si al principio no había conseguido encontrar dos celdas que formasen pareja, utilizaría un “replicar”. De esta manera, volvería a buscar parejas y todo lo que ello conlleva.

El juego automático acaba en el momento en el que se le acaban las “réplicas” o no hay espacio para “replicar” o gane.

3. Planificación y tareas

Programador	Horas	Tarea
Ambos	2	Lectura de la documentación inicial, planificación del trabajo en grupo.
Ambos	0.5	Prueba del proyecto base
Ambos	1	Diseño general de la aplicación y de los TAD necesarios
Ambos	2	Diseño e implementación del TAD Celda (estructura de datos, implementación de operaciones y pruebas)
Ambos	0.5	Definición y diseño del TAD Tablero (estructura de datos y operaciones necesarias)
Ambos	10	Implementación del TAD Tablero (juegos de pruebas y operaciones)
Ambos	7	Definición e implementación del TAD juego (estructura de datos y operaciones)
Ambos	3	Pruebas generalizadas del proyecto
Ambos	5	Ampliaciones del proyecto: Ampliación 1.
Ambos	1	Ampliaciones del proyecto: Ampliación 2.
Ambos	1	Ampliaciones del proyecto: Ampliación 3.
Ambos	4	Escritura final de la documentación (una parte de esta tarea se debe realizar a lo largo de todo el proceso, mientras se implementan los TAD)

Total: 37 horas (cada alumno).

El proyecto se ha realizado en **37 horas**, con el uso de **sesiones de código compartidas en tiempo real** mediante *CodeTogether*, con una cantidad aproximada de **8 sesiones** de varias horas.

El único **acuerdo** al que se ha llegado con respecto al ámbito del desarrollo del programa ha sido que **Manuel Alonso programase el módulo replicar** del TAD Tablero de manera individual, con ayuda de las **observaciones de José Manuel de Torres.**

4. Conclusiones y principales problemas

El principal problema se dio con el módulo replicar, dado que era el módulo más complejo de todo el proyecto bajo nuestro punto de vista. Para solucionar este problema, y con el motivo de trabajar de forma sencilla y eficiente, optamos por que este módulo se realizase por una persona del equipo, ya que así no había confusiones entre ambos, dado que proponíamos métodos diferentes. Manuel Alonso se ofreció a hacerlo, con las observaciones de José Manuel de Torres.

En general, el desarrollo del proyecto se ha llevado de una manera fluida y con problemas menores. Cabe destacar la participación del hermano de Manuel Alonso, que jugó al juego para que pudiésemos ver si había algún problema de implementación.

En cuanto al ámbito de la programación, se recurrió a una serie de herramientas externas que ayudaron al desarrollo, limpieza y mantenimiento del proyecto:

- *CodeTogether*, una extensión de Eclipse enfocada en la colaboración y edición de código en tiempo real, que nos permitió la edición de código desde dos dispositivos distintos sobre varios archivos al mismo tiempo. *CodeTogether* ofrece una versión gratuita limitada como prueba limitada de sus servicios.

  Sin embargo, conseguimos una licencia de pago por pertenecer a la Universidad de Extremadura, por lo que pudimos trabajar juntos sin necesidad de acceder a funciones limitadas o estar limitados por sesiones de 1 hora como máximo.

- *FreeFileSync*, un programa de gestión de copias de seguridad, nos permitió poder guardar y mantener sincronizados bidireccionalmente los archivos del proyecto entre la máquina virtual y la máquina huésped (Windows) mediante el uso del almacenamiento de las cuentas de Google Drive de la UEx, sin tener que usar carpetas bidireccionales o dispositivos USB virtualizados.

  De esta manera, los archivos quedaban guardados localmente y en la nube entre Google Drive, la máquina virtual y la máquina huésped.

- El *cerebro*, una herramienta potente, aunque bastamente inutilizada por la juventud de hoy en día.