Instituto Tecnológico de Costa Rica

Sede Central Cartago

Escuela de Ingeniería en Computación

Programación Orientada a Objetos, IC-2101

Programa 3

"Mantenimiento de software-Futoshiki"

Estudiantes:

Brandon Andrés Mora Díaz, 2022164409

Natalia Sofía Rodríguez Solano, 2021033307

GR-2

Docente:

William Mata Rodríguez

II Semestre

2022

Tabla de Contenidos

Enunciado del proyecto 3

Temas investigados 8

Patrón MVC 8

Solución 10

Conclusiones 11

Referencias bibliográficas 12

Lista de Revisión del Proyecto 13

Enunciado del proyecto

PROYECTO: MANTENIMIENTO DE SOFTWARE

MEJORAS AL PROGRAMA FUTOSHIKI

DEFINICIÓN DEL PROYECTO

En la ingeniería de software el ciclo de vida de desarrollo de software (SDLC: System Development Life Cycle) se refiere al conjunto de etapas necesarias para la creación y utilización de software a través del tiempo.

Hay diferentes metodologías que soportan este ciclo de vida, algunas de ellas son: modelo en cascada (el más antiguo), modelo en espiral, prototipos, modelo incremental e iterativo, desarrollo ágil (usando herramientas como SCRUM, programación extrema-XP, Kanban, etc.).

Dentro de las metodologías de desarrollo está la etapa de mantenimiento de software.

El mantenimiento de software es una actividad natural del ciclo de vida del software, es muy común que nos encontremos trabajando en ello. Consiste en modificar el software por diversos motivos:

- Corrección de errores
- Adaptar el software a nuevos requerimientos
- Mejoras

En este proyecto vamos a hacer mantenimiento al software realizado en el proyecto anterior: específicamente vamos a hacer mejoras de dos tipos:

- A nivel técnico: se implementará el patrón de arquitectura MVC (Modelo/Vista/Controlador)
- A nivel funcional: se agregarán nuevas funcionalidades.

REQUERIMIENTOS DEL PROGRAMA: NUEVAS FUNCIONALIDADES

1- Tamaño de la cuadrícula

La versión actual maneja una cuadrícula de tamaño 5, esta nueva funcionalidad agrega cuadrículas de tamaño 6, 7, 8 y 9. Este cambio modifica la estructura del archivo de partidas: se le agrega el dato de cuadrícula según puede observar en la definición del archivo PARTIDAS DE JUEGO. También cambian las partes donde se trata el tamaño de la cuadrícula ya que pasa de tamaño 5 a tamaños 5, 6, 7, 8 y 9.

Para desarrollar esta funcionalidad hay que cambiar la opción de Configurar:

Agregar el dato 4. Tamaño de la cuadrícula.

2- Juegos multinivel

El objetivo de esta funcionalidad es que el programa pueda ir avanzado automáticamente al jugador en los niveles de juego: el jugador empieza a jugar en el nivel fácil. Cuando logre terminar exitosamente un juego en este nivel, el programa automáticamente lo envía a jugar al nivel intermedio. Cuando logre terminar exitosamente un juego en este nivel, el programa automáticamente lo envía a jugar al nivel difícil. En cada nivel el programa determinará si va al Top-10 correspondiente. Cuando termina el último nivel se queda jugando en dicho nivel. Cuando se juega con reloj o timer, este abarca el tiempo utilizado en todos los niveles.

Por ejemplo: si hay un timer de 1 hora quiere decir que esa hora es para completar todos los niveles.

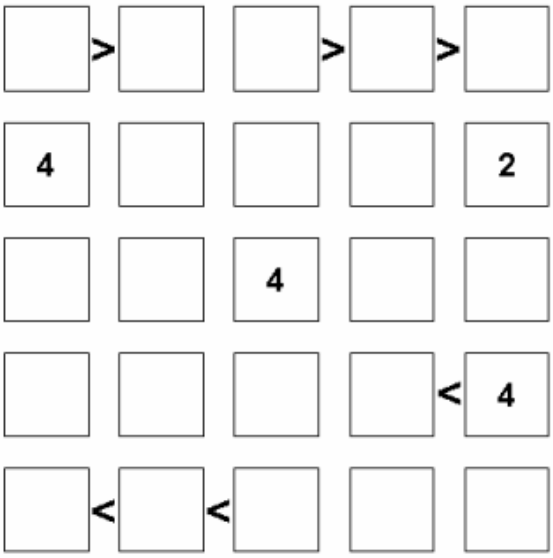
Para desarrollar esta funcionalidad hay que cambiar la opción de Configurar:

- En Nivel: agregar la opción de Multinivel

3- Posibles jugadas para una casilla.

Esta funcionalidad sirve para desplegar al usuario todas las posibles jugadas correctas (dígitos) que puede hacer en ese momento para una casilla específica. Por ejemplo, si tenemos el siguiente juego y queremos saber las posibles jugadas para la casilla en la fila 3 columna 5 hay que desplegar esta información: 1, 3, 5

.



Los cálculos de los posibles dígitos deben ser dinámicos, es decir, se calculan en el momento considerando el juego que se tiene.

Decida usted la interfaz de usuario para implementar esta funcionalidad.

Opción Configurar

Esta opción es para indicar las condiciones con que se va a jugar. Contiene los siguientes datos que se van a guardar en el archivo "futoshiki2022configuración.dat" : (los valores por omisión –o default- están señalados con el círculo en rojo)

1. Nivel:
   - o Fácil
   - o Intermedio
   - o Difícil
   - o Multinivel
2. Reloj:
   - o Si
   - o No
   - o Timer

Horas	Minutos	Segundos
0	30	0

Restricciones de los datos para el timer: las horas pueden estar entre 0 y 2, los minutos entre 0 y 59 y los segundos entre 0 y 59. El timer debe tener al menos uno de estos valores. Hay que realizar estas validaciones y enviar los mensajes respectivos en caso de errores.

La medición del tiempo es tiempo real.

3. Posición en la ventana del panel de dígitos:
   - o Derecha
   - o Izquierda
4. Tamaño de la cuadrícula:
   - o 5
   - o 6
   - o 7
   - o 8
   - o 9

PARTIDAS DEL JUEGO

Registre partidas de este juego en el archivo "futoshiki2022partidas.xml". Busque algunas partidas, al menos 3 por cada nivel, y grábelas directamente en el archivo, fuera del programa del juego.

Estructura del archivo con formato XML:

<partidasFutoshiki>

< partida >

<nivel>nivelDificultad</nivel>

<cuadricula>numero</cuadricula>

<desigualdades>>tipoDesigualdad, indiceFila, indiceColumna, …

</desigualdades>

<constantes>constante, indiceFila, indiceColumna, …

</constantes>

< /partida >

</partidasFutoshiki>

nivelDificultad: Facil, Intermedio, Dificil

numero: 5, 6, 7, 8, 9

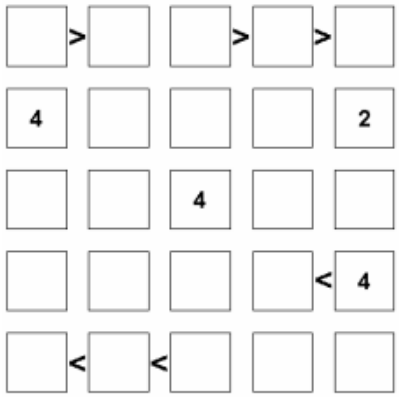
tipoDesigualdad: a (>), b (<), y (v), z (^),

índice de fila, índice de columna: de casilla que está a la izquierda para desigualdad de fila o de casilla superior para desigualdad de columna

constante: entero entre 1 y el tamaño de la cuadrícula

índice de fila, índice de columna de la constante

Ejemplo con esta partida de nivel fácil:




```
<partidasFutoshiki>
        <partida>
                <nivel>Facil</nivel>
                <cuadricula>5</cuadricula>
                <des>a,0,0</des>
                <des>a,0,2</des>
                <des>a,0,3</des>
                <des>b,3,3</des>
                <des>b,4,0</des>
                <des>b,4,1</des>
                <cons>4,1,0</cons>
                <cons>2,1,4</cons>
                <cons>4,2,2</cons>
                <cons>4,3,4</cons>
        </partida>
</partidasFutoshiki>
```

DOCUMENTACIÓN DEL PROYECTO

Trabajo en grupos de 2 personas máximo. Se mantienen los mismos grupos que trabajaron en el proyecto 2 ya que son cambios a dicho trabajo.

Se coordinará un día y hora para revisar el proyecto junto con el estudiante, quien siendo su autor debe demostrar un completo dominio de la solución implementada tanto desde el punto de vista técnico como de la funcionalidad (lo que hace la solución). En la revisión se pueden realizar estas actividades:

- Revisar esta solución particular
- Revisar conceptos incluidos en la evaluación
- Aplicar otras actividades con una complejidad igual o menor a la evaluación.

REQUISITOS PARA REVISAR EL PROYECTO

a- El programa debe tener documentación interna y usar JavaDoc como parte de esa documentación.
b- La nota de la documentación del proyecto indicada abajo sirve para aceptar o rechazar el proyecto: se revisan los proyectos que cumplan con esa documentación en un 90% o más.
c- Desarrollo en Java usando GUI.
d- Opcionalmente pueden usar software de control de versiones y trabajo colaborativo (por ejemplo, github)

Enviar vía tecDigital, sección EVALUACIONES / PROGRAMAS, una carpeta comprimida (nombre **programa3_sus_nombres.zip**) que contenga la siguiente documentación (nombre **programa3_documentación_del_proyecto.PDF**) y la carpeta de desarrollo del proyecto (nombre **programa3_futoshikiV2**):

- Portada. (1P)
- Contenido de la documentación. (2P)
- Enunciado del proyecto -esta especificación-. (2P)
- Temas investigados (material no estudiado en el curso) (0P o 35P). Por cada uno de estos temas debe poner el marco teórico: de qué trata, cómo se usa. Omita la parte relacionada con el uso de interfaces gráficas. Al menos deben investigar:

  o El patrón de arquitectura MVC (Modelo/Vista/Controlador)

- Solución (0P o 20P)

  o Modelo del sistema con un diagrama de clases UML que incluya (actualizar modelo con las mejoras):

  ▪ Atributos sus tipos de datos

  ▪ Métodos, incluyendo setters/getters: usar nombres como setIdentificacion(String pIdentificacion, getIdentificacion(). Incluya tipos de datos, parámetros, valores de retorno.

  ▪ Relaciones entre los objetos de las clases: dependencia, asociación, agregación, composición, herencia ▪ Navegabilidad, multiplicidad

  ▪ Opcionalmente nombre de asociaciones y roles

  En el diagrama no presente atributos ni estructuras de datos que soportan la implementación de las relaciones, el diagrama al mostrar las relaciones muestra esos aspectos que luego se implementarán. Los constructores de cada clase deben ser con parámetros. No se permiten componentes duplicados.

Temas investigados

Patrón MVC

un patrón MVC corresponde a un patrón arquitectural, un modelo o guía que expresa cómo organizar y estructurar los componentes de un sistema software, sus responsabilidades y las relaciones existentes entre cada uno de ellos.

Su nombre viene de sus iniciales las cuales corresponden a Modelo-Vista-Controlador (Model-View-Controller, en inglés), que son las capas o grupos de componentes en los que se pueden organizar aplicaciones bajo este paradigma. Separa la lógica de la aplicación de la lógica de la vista en una aplicación. Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales.

En este tipo de arquitectura existe un sistema central o controlador que gestiona las entradas y la salida del sistema, uno o varios modelos que se encargan de buscar los datos e información necesaria y una interfaz que muestra los resultados al usuario final.

Esta estructura es muy utilizada en el desarrollo web porque al tener que interactuar varios lenguajes para crear un sitio es muy fácil generar confusión entre cada componente si estos no son separados de la forma adecuada. Este patrón permite modificar cada uno de sus componentes si necesidad de afectar a los demás.

Modelo

La primera capa corresponde a la de **Modelo**; en este se encontrará una presentación de los datos del dominio, es decir, aquellas entidades que servirán para almacenar información del sistema que se quiere desarrollar; igual manera su lógica de negocio, y sus mecanismos de persistencia.

Esta capa es responsable de:

- ✓ Acceder a la capa de almacenamiento de datos; idealmente el modelo debe ser independiente del sistema de almacenamiento.
- ✓ Definir las reglas de negocio (la funcionalidad del sistema).
- ✓ Lleva un registro de las vistas y controladores del sistema.
- ✓ En caso de estar ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo.

Vista

La segunda capa a explicar es la de **Vista**. Los componentes de la son los responsables de generar la interfaz en una aplicación, es decir, de componer las pantallas, páginas, o cualquier tipo de resultado utilizable por el usuario o cliente del sistema.

Cuando estas componen la interfaz de usuario de una aplicación, debe de contener elementos de interacción que permitan al usuario enviar información e invocar acciones en el sistema, como lo son los botones, cuadros de edición o cualquier otro tipo de elemento, conveniente que se adapte a la tecnología del cliente.

Esta capa es responsable de:

- ✓ Recibir datos del modelo y los muestra al usuario.
- ✓ Tener un registro de su controlador asociado (normalmente porque además lo instancia).

- ✓ Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

Controlador

La última capa a explicar corresponde a el **Controlador**. La principal función de los componentes existentes en esta capa es actuar como intermediarios entre el usuario y el sistema, es decir deberán capturar las acciones de este sobre la Vista, como puede ser la pulsación de un botón o la selección de una opción de menú, interpretarlas y actuar en función de ellas. De igual forma tendrá que

Esta capa es responsable de:

- ✓ Recibir los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
- ✓ Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas; una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener_tiempo_de_entrega (nueva_orden_de_venta)" por ejemplo.

Este patrón es uno de los más utilizados. En la actualidad se puede encontrar tanto en pequeños como en grandes sistemas, en el mundo laboral es indispensable llevarlo a la práctica.

Algunas de las ventajas y desventajas de este patrón estructural son las siguientes:

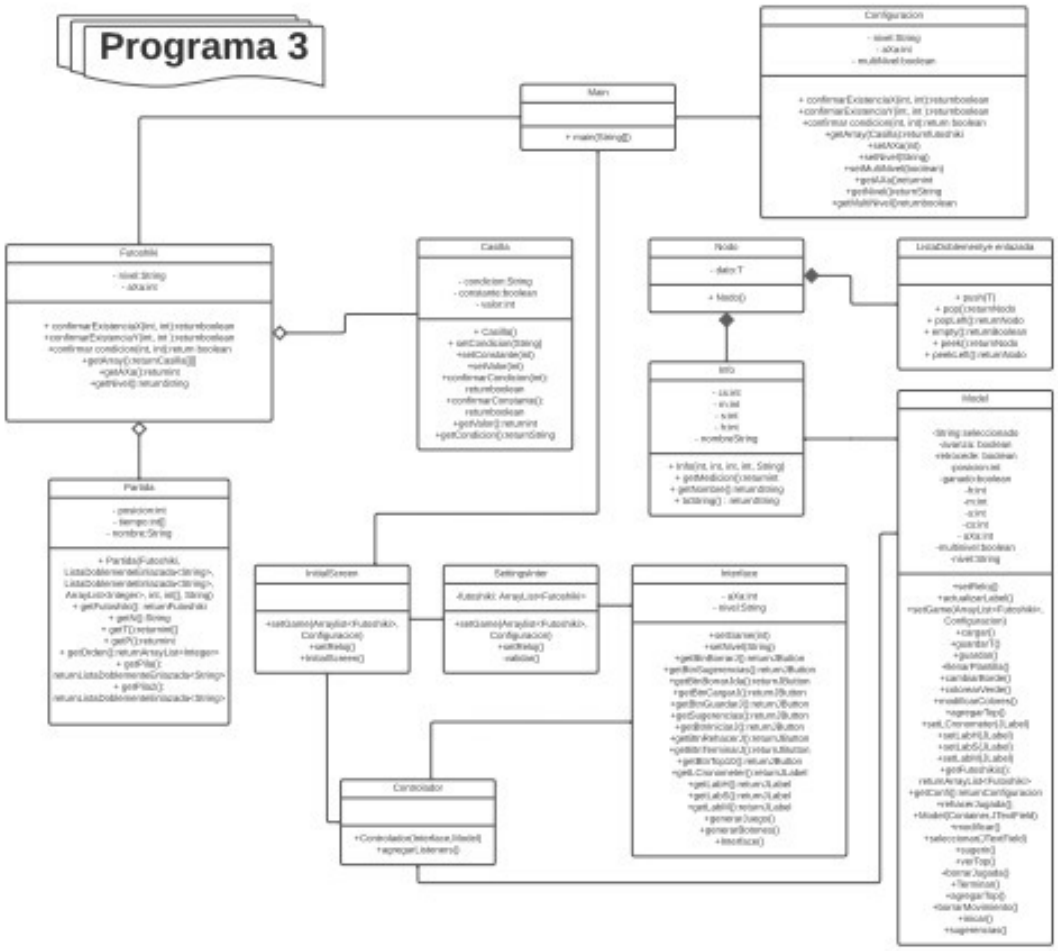
Ventajas

- ✓ Tiene una fácil organización, puesto que solo cuenta con tres componentes.
- ✓ Es un patrón que se puede adaptar a diferentes frameworks.
- ✓ Se puede escalar fácilmente.
- ✓ Facilita el trabajo en equipo.

Desventajas

- ➢ Puede ser un patrón complicado de aprender.
- ➢ No se suele usar en programas sencillos.
- ➢ Requiere una gran cantidad de ficheros o carpetas.

Solución



Enlace del Diagrama UML realizado:

https://lucid.app/lucidchart/70ade985-a16a-4f75-99cb-1ca4c7322b9d/edit?viewport_loc=-1465%2C-569%2C5232%2C2373%2C0_0&invitationId=inv_56f8656b-8bbd-4bf9-88bd-1ea2646fc693

Conclusiones

En el presente trabajo se logró estudiar, comprender y aplicar el patrón estructural que corresponde al MVC, como arquitectura que separa la lógica de la aplicación de la lógica de la vista en una aplicación, siendo importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas de alto nivel; se logró reforzar y poner en práctica el manejo de los datos y paso de parámetros para la conexión de las ventanas y sus debidos requerimientos.

De la misma forma se cumplió el objetivo principal del trabajo enfocándose en el mantenimiento, adaptación del software, corrección de errores y mejora de un programa ya hecho; se lograron cambiar aspectos los cuales fueron más eficientes para la ejecución de dicho programa.

Referencias bibliográficas

Aguilar J. (2019) ¿Qué es el patrón MVC en programación y por qué es útil? Campus MVP. Recuperado de: https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx

Universidad de Alicante (2022). Modelo vista controlador (MVC). Universidad de Alicante. Recuperado de: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

Hernández U. (2015). MVC (Model, View, Controller) explicado. Código facilito. Recuperado de: https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

García M. (2017). MVC (Modelo-Vista-Controlador): ¿qué es y para qué sirve? Coding or not. Recuperado de: https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve

KeepCoding (2022). Qué es la arquitectura MVC. KeepCoding. Recuperado de: https://keepcoding.io/blog/que-es-la-arquitectura-mvc/

Lista de Revisión del Proyecto

Concepto	Puntos	Avance 100/%/0	Puntos obtenidos	Análisis de resultados
Juegos multinivel (6 puntos por cada nivel)	18	100%	18	
Posibles jugadas para una casilla	12	100%	12	
Cambios: Opción Configurar Ventana del juego Grabar juego Cargar juego Archivo Partidas del juego	 10 10 5 5 5	100%	 10 10 5 5 5	
Aplicación correcta del patrón de arquitectura MVC	25	100%	25	
Ayuda (Desplegar achivo con la documentación actualizada)	10	100%	10	
TOTAL	**100**	100%	100	
Partes desarrolladas adicionalmente				