Escuela de Ingeniería de Computación

IC-1803 Taller de Programación

Proyecto Programado #3

FUTOSHIKI

José Ricardo Cardona Quesada / Carné: 2021022613

Profesor: William Mata Rodríguez

Fecha de Entrega: 29 de junio, I Semestre 2021

Link del repertorio: https://github.com/JoseCardonaQ/futoshiki

Tabla de Contenidos

Descripción del Proyecto 3
Temas Investigados 14
Radio buttons 14
Función After 15
Importancia del Software de control de Versiones 15
Git (Software de control de versiones usado) 16
Conclusiones 20
Diseño y Explicación de la Solución 21
Configuración 21
Jugar 22
Seleccionar Partida 22
Desplegar una partida 23
Darle un valor a un botón 24
Iniciar Juego 24
Borrar Jugada 25
Terminar Juego y Borrar Juego 25
Guardar y Cargar Juego 26
Top10 27
Estadística de tiempos 28
REVISIÓN DEL PROYECTO 33
Bibliografía 34

zz

Descripción del Proyecto

Instituto Tecnológico de Costa Rica - Escuela de Computación
Carrera: Ingeniería en Computación - Curso: Taller de Programación – Semestre: I 2021
Programa 3 (30%): FUTOSHIKI
Fecha de entrega: martes 29 de junio, 8 A.M. (8 DE LA MAÑANA) / Revisión: 29 y 30 de junio

DEFINICIÓN DEL PROYECTO: FUTOSHIKI

Futoshiki es un pasatiempo de lógica originario de Japón, desarrollado por Tamaki Seto en 2001. Su nombre significa "desigualdad". La información de este juego se tomó de wikipedia (https://en.wikipedia.org/wiki/Futoshiki). Puede jugarlo en línea en https://es.goobix.com/juegos-en-linea/futoshiki/.

En este pasatiempo hay que llenar con dígitos las casillas de una cuadrícula de tal forma que cada dígito no se repita ni en la fila ni en la columna a que pertenece y los dígitos cumplan con restricciones de desigualdad: mayor que (>) o menor que (<). Las restricciones de desigualdad pueden estar a nivel de filas (entre dos casillas horizontales) o a nivel de columnas (entre dos casillas verticales). La cuadrícula debe tener la misma cantidad de filas y de columnas.

En caso de que la cuadrícula fuera de tamaño 5 x 5, los dígitos a colocar estarían entre 1 y el tamaño de la cuadrícula, en este caso 5. Cada juego tiene de forma predeterminada las restricciones de desigualdad que el jugador debe cumplir y algunos dígitos fijos como se muestra en la figura 1:

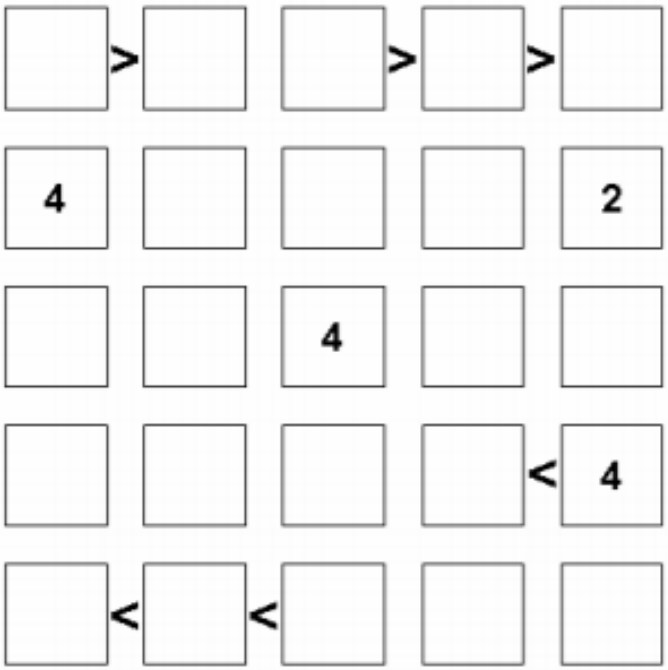


Figura 1: ejemplo de un Futoshiki 5 x 5 (By Gandalf61 at the English Wikipedia / CC BY-SA, http://creativecommons.org/licenses/by-sa/3.0/)

La solución al juego anterior sería:

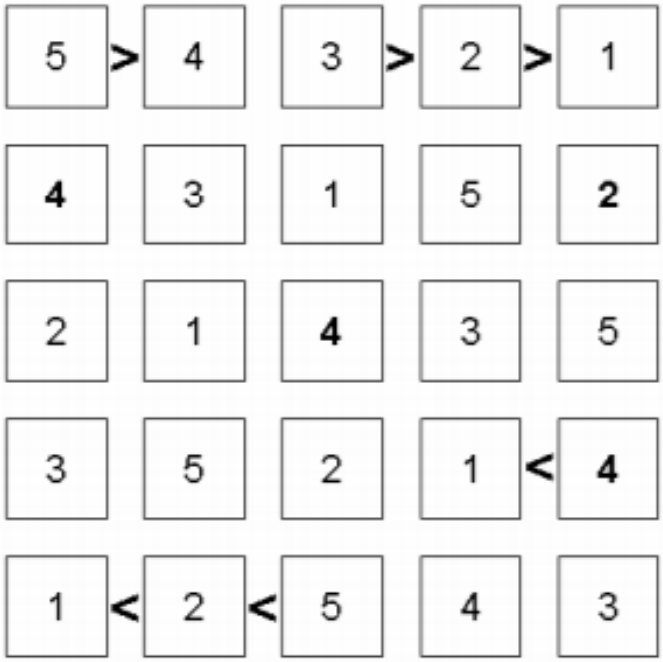
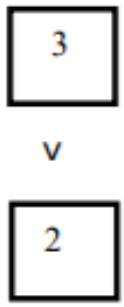
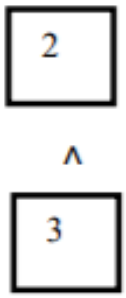


Figura 2: Futoshiki terminado (By Gandalf61 at the English Wikipedia / CC BY-SA, http://creativecommons.org/licenses/by-sa/3.0/).

El juego puede tener restricciones a nivel de columna, por ejemplo:



Se lee: 3 > 2



Se lee: 2 < 3

El programa usará una GUI (Graphical User Interface) que inicia con un menú principal desde el cual se accederá la funcionalidad del programa, es decir, lo que el programa hace. Usted puede agregar otras funcionalidades que vayan a mejorar el producto. Puede hacer cambios a la interfaz previamente acordados con el profesor.

Objetivos del proyecto de programación:

- Aplicar el ciclo completo de la metodología general de desarrollo de programas a situaciones de mayor alcance.

- Aplicar y reforzar conceptos de programación y del lenguaje Python 3.
  - Uso de diversos componentes del lenguaje.
  - Desarrollo y reutilización de funciones.
  - Uso de estructuras condicionales y de repetición de procesos. Opcionalmente puede usar recursión en las partes que considere apropiadas.
  - Diseño y uso de estructuras de datos nativas de Python.
  - Diseño y uso de TDA (Tipos de Datos Abstractos).
  - Uso de archivos.

- Aplicar buenas prácticas de programación: documentación interna y externa del programa, reutilización de código, nombres significativos, eficiencia del programa, evaluar alternativas, uso de técnica divide y vencerás (dividir el problema en partes, desarrollar cada una de esas partes), etc.

- Validación de los datos de entrada: todos los datos de entrada se deben validar según restricciones que se indican en cada uno de ellos.

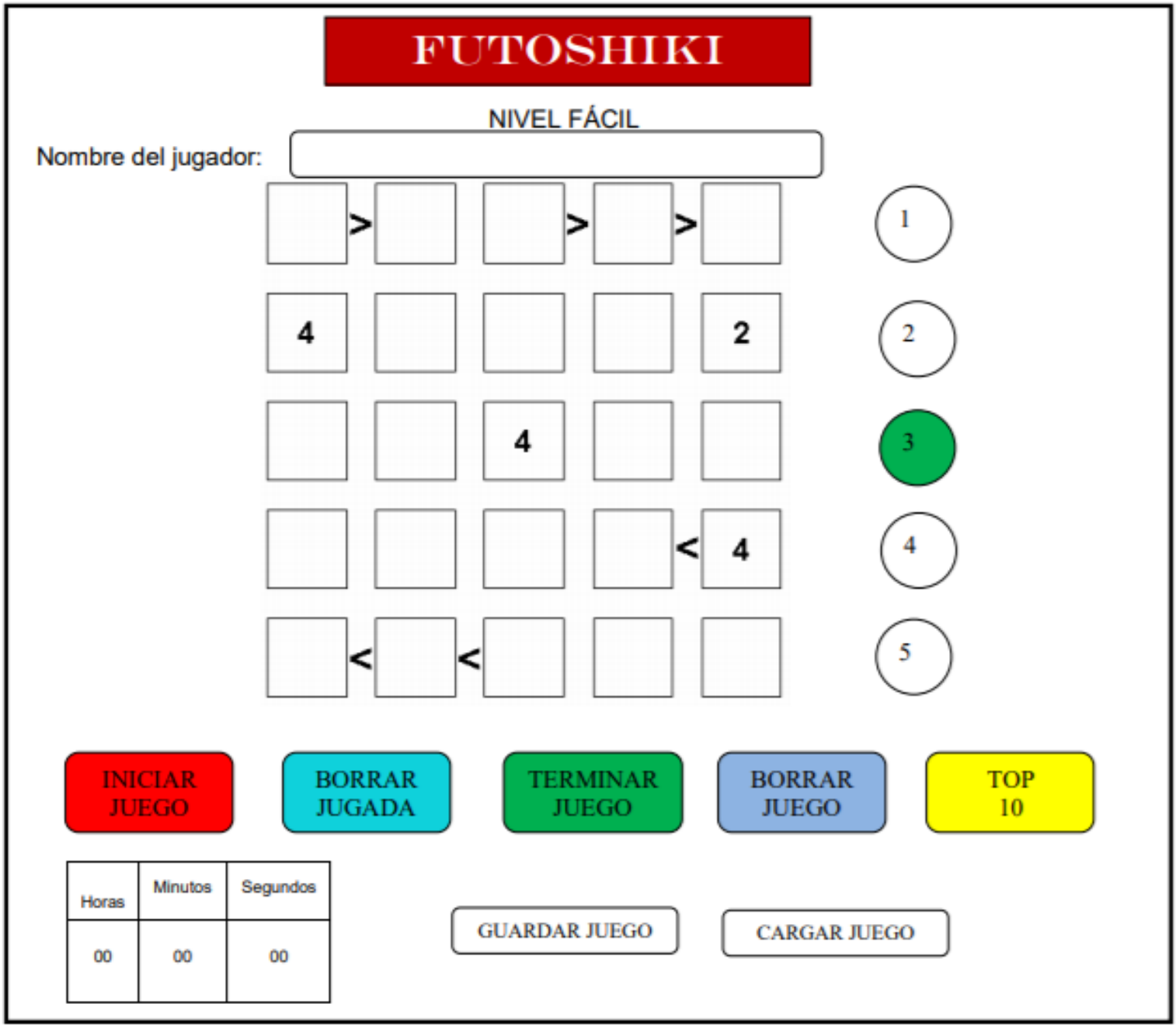
- Fomentar la investigación en el estudiante: aquellos temas no tratados en el curso pero que los necesita para hacer el proyecto. Dichos temas deben ser explicados detalladamente en la documentación del proyecto. Entre los tópicos a investigar para este proyecto específico están:
  - Nuevas características de tkinter exploradas para este proyecto.
  - Software de control de versiones: importancia en ingeniería del software.
  - Software de control de versiones usado: Git u otro, explicar las funciones usadas.
  - Cualquier otro aspecto necesario para ofrecer su solución.

- **Usar algún software de control de versiones de software, por ejemplo, Git u otro que usted decida. EN LA SEMANA SE DARÁ UN TALLER DEL SOFTWARE GIT HUB.**

REQUERIMIENTOS DEL PROGRAMA

A) Jugar

Esta opción permite jugar el Futoshiki con un tamaño de cuadrícula 5 x 5. Cuando se da esta opción se muestra una pantalla como la siguiente considerando la configuración (nivel del juego, uso del reloj, posición del panel de dígitos).



El programa tiene una serie de partidas que previamente han sido registradas y de ahí selecciona aleatoriamente una partida según el nivel de dificultad configurado. Python tiene

funciones para generar números aleatorios que pueden servir para seleccionar alguna de las partidas de tal forma que siempre se elija una al azar. Puede usar otros algoritmos para esta selección aleatoria de partidas. En el diseño de la solución documente el algoritmo de selección aleatoria usado aquí.

Uso de los botones:

Cuando el jugador pica este botón se inicia el juego. Una vez que se inicia el juego, el jugador selecciona un dígito picándolo en el panel de dígitos (círculos a la derecha de la partida) y luego pica en la casilla de la cuadrícula en donde quiere ponerlo. Cuando pica el dígito se marca su círculo con un color verde como se muestra en el ejemplo. El dígito permanece marcado, disponible para ponerlo en otra casilla. Si quiere seleccionar otro dígito lo pica y el dígito anterior se desmarca. Cuando el jugador pone un dígito se deben hacer las validaciones para que la jugada cumpla con las reglas del juego, de lo contrario se le envía alguno de estos mensajes:

- JUGADA NO ES VÁLIDA PORQUE EL ELEMENTO YA ESTÁ EN LA FILA
- JUGADA NO ES VÁLIDA PORQUE EL ELEMENTO YA ESTÁ EN LA COLUMNA
- JUGADA NO ES VÁLIDA PORQUE NO CUMPLE CON LA RESTRICCIÓN DE MAYOR
- JUGADA NO ES VÁLIDA PORQUE NO CUMPLE CON LA RESTRICCIÓN DE MENOR
- JUGADA NO ES VÁLIDA PORQUE ESTE ES UN DÍGITO FIJO

  En estos casos ponga en color rojo la casilla que no está permitiendo la jugada para que el jugador la identifique de inmediato. El mensaje es enviado y el programa se detiene para que el jugador vea ese mensaje, cuando quiera seguir jugando debe dar la tecla <ENTER> u otra tecla y se quita el color rojo de la casilla con error.

El juego termina cuando el jugador llena todas las casillas de la cuadrícula de forma correcta. En este caso el reloj o el timer se detienen automáticamente y se despliega un mensaje de felicitación, por ejemplo: ¡EXCELENTE! JUEGO TERMINADO CON ÉXITO.

En este momento debe determinar si este jugador debe registrarlo en el Top 10. El Top 10 es un archivo donde el programa registra las mejores 10 marcas por cada nivel de dificultad (los jugadores que tarden menos en completar el juego). Si tenemos las 10 marcas y el jugador actual hace un mejor tiempo que esas marcas, hay que eliminar la marca con mayor tiempo para seguir teniendo un máximo de 10 marcas por nivel. La marca contiene el nombre del jugador y el tiempo (horas, minutos, segundos) que un jugador tardó en completar un

juego. Note que si usa el timer hay que calcular la duración del juego. Cuando un juego es terminado con éxito el programa regresa a la opción de Jugar.

Otras consideraciones:

- Antes de iniciar el juego el jugador debe dar un nombre (string de 1 a 20 caracteres). Este nombre no debe estar en el TOP 10 de cualquier nivel.
- Luego de dar el botón INICIAR JUEGO, este botón se deshabilita.
- En caso de haber configurado la opción de Timer, el jugador puede dejar el tiempo registrado en la configuración o modificarlo antes de INICIAR JUEGO. El tiempo empieza a correr cuando le den INICIAR JUEGO.
- En caso de no usar el reloj o el timer no debe aparecer esa parte en la pantalla. Para el uso del timer alguna de sus partes (horas, minutos, segundos) debe ser mayor a cero.
- En caso de haber configurado la opción de Timer y éste llegue a 0 y el juego no haya terminado se envía el mensaje TIEMPO EXPIRADO. ¿DESEA CONTINUAR EL MISMO JUEGO (SI O NO)?. Si responde SI entonces el timer pasa a ser reloj inicializado con el tiempo que se había establecido en el timer. Por ejemplo si el timer estaba para 1 hora y 30 minutos, ahora el reloj debe marcar que ya ha pasado 1 hora y 30 minutos y sigue contando el tiempo. Si responde NO el juego finaliza regresando a la opción de Jugar.
- En caso de usar el reloj hay que validar los valores dados de horas, minutos y segundos. En la configuración están los rangos de estos datos.
- En caso de no existir alguna partida para el nivel seleccionado se da el mensaje NO HAY PARTIDAS PARA ESTE NIVEL. Luego lo envía al menú principal.
- En caso de picar una casilla y no haya seleccionado previamente un dígito se envía el error FALTA QUE SELECCIONE UN DÍGITO.

En caso de que ocurra un error el programa enviará un mensaje respectivo. El error se mantendrá en pantalla hasta que el usuario presione la tecla <Enter> para continuar. Con esto se pretende que el usuario vea el mensaje de error y luego decida cuando continuar.

BORRAR JUGADA

Elimina la última jugada dejando la casilla vacía. Puede borrar todas las jugadas que ha hecho. Use un TDA pila para registrar la información de las jugadas que van sucediento de tal manera que pueda implementar esta funcionalidad. Cada vez que se hace una jugada se agrega a la pila (fila y columna de la jugada), y si seleccionan este botón, se toma la última jugada agregada en la pila (el manejo de la pila es tipo LIFO: Last In First Out, último en

entrar primero en salir) y se borra la casilla respectiva de la cuadrícula. La jugada se quita de la pila. En el diseño de la solución describa la estructura del TDA usado y el algoritmo de manipulación.

Otras consideraciones:

- En caso de que ya no hayan más jugadas para borrar según la pila hay que enviar el mensaje NO HAY MÁS JUGADAS PARA BORRAR.
- Se puede seleccionar esta opción solamente si el juego ha iniciado, antes de ello la opción permanece deshabilitada.
- La pila de jugadas se reinicia en cada juego.

TERMINAR JUEGO

Cuando el jugador selecciona esta opción se le pregunta

¿ESTÁ SEGURO DE TERMINAR EL JUEGO (SI o NO)?

Si responde SI termina de inmediato el juego y se vuelve a mostrar otro juego como si estuviera entrando a la opción de Jugar .

Si responde NO sigue jugando con el mismo juego.

Se puede seleccionar esta opción solamente si el juego ha iniciado, antes de ello la opción permanece deshabilitada.

BORRAR JUEGO

Cuando el jugador selecciona esta opción se le pregunta

¿ESTÁ SEGURO DE BORRAR EL JUEGO (SI o NO)

Si responde SI vuelve a la opción de Jugar usando la misma partida pero eliminando todas las jugadas que hizo.

Si responde NO sigue jugando con el mismo juego.

Se puede seleccionar esta opción solamente si el juego ha iniciado, antes de ello la opción permanece deshabilitada.

TOP
10

Esta opción se puede usar en cualquier momento. Detiene el reloj si lo está usando. Despliega una sola pantalla con los registros de los mejores 10 primeros jugadores por cada nivel: aquellos que hicieron menos tiempo para completar el juego. En caso de no tener los 10 jugadores en algún nivel se despliegan los que se tengan. El Top 10 se guarda en el archivo "futoshiki2021top10.dat".

Represente el Top 10 con 3 listas, una por cada nivel de dificultad, los elementos de cada lista son tuplas con la información del registro.

TOP 10

NIVEL DIFÍCIL:	*JUGADOR*	*TIEMPO*
1-	Nombre jugador	1:30:15
2-	Nombre jugador	1:32:55
	…	
10-		

NIVEL INTERMEDIO:	*JUGADOR*	*TIEMPO*
1-	Nombre jugador	1:10:21
2-	Nombre jugador	1:35:55
	…	
10-		

NIVEL FÁCIL:	*JUGADOR*	*TIEMPO*
1-	Nombre jugador	0:7:23
2-	Nombre jugador	0:10:55
	…	
10-		

Luego de que el usuario vea esta información el programa regresa a donde estaba jugando y sigue el conteo en el reloj.

GUARDAR JUEGO

Este botón se puede usar en cualquier momento que el juego haya iniciado, antes de ello permanece inhabilitado. Guarda en el archivo "futoshiki2021juegoactual.dat" todo el estado del juego actual: configuración, cuadrícula, nombre del jugador, etc. El objetivo es que el jugador pueda en cualquier momento guardar el juego y posteriormente continuarlo en el punto donde hizo el guardado del juego. Este archivo solo va a contener una partida. En caso de que haya una partida en el archivo, se borra y se guarda la del momento.

CARGAR JUEGO

Este botón se puede usar solamente cuando un juego no se haya iniciado, luego de ello permanece inhabilitado. Trae del archivo "futoshiki2021juegoactual.dat" el juego que fue guardado y lo pone en la pantalla como el juego actual con exactamente el mismo estado que tenía cuando fue guardado. El juego continúa cuando el jugador usa el botón de INICIAR JUEGO.

B) 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 "futoshiki2021configuración.dat" : (los valores por omisión –o default- están señalados con el círculo en rojo)

1. Nivel: ● Fácil
   o Intermedio
   o Difícil

2. Reloj: ● Si
   o No
   o Timer

Horas	Minutos	Segundos
0	30	0

   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: ● Derecha
   o Izquierda

C) Ayuda

Esta opción la usaremos para que el usuario pueda ver el Manual de Usuario directamente en la computadora (despliega el pdf respectivo).

Acerca de

Puede poner esta opción para desplegar información "Acerca del programa" donde pondremos al menos los datos del nombre del programa, la versión, la fecha de creación y el autor.

Salir

Puede poner esta opción para salir del programa. También se puede salir con la opción de cerrar "X" en la GUI.

PARTIDAS DEL JUEGO

Registre partidas de este juego en el archivo "futoshiki2021partidas.dat". Las partidas se van a almacenar en tres listas:

- Lista de partidas de nivel fácil
- Lista de partidas de nivel intermedio
- Lista de partidas de nivel difícil

Cada elemento de estas listas representa una partida por medio de una tupla. Y cada partida se componen de tuplas conteniendo una restricción o un dígito fijo de la partida.

Estructura de una tupla con restricciones:
(tipo de restricción: para filas ">" o <"; para columnas "v" o "^",
posición de la casilla izquierda o superior de la desigualdad: índice de fila, índice de columna)

Estructura de una tupla con dígitos fijos:
(dígito: "1", "2", "3",...,
posición del dígito: índice de fila, índice de columna)

Con esta partida de ejemplo de nivel fácil:

	>		>	>
4				2
		4		
			<	4
<	<			

tendríamos la siguiente estructura:

```
[           # lista de partidas de nivel fácil:
        (               # tupla de la partida número 1
        (">", 0, 0), (">", 0, 2), (">", 0, 3),      # fila 1
        ("4", 1, 0), ("2", 1, 4),                   # fila 2
        ("4", 2, 2),                                # fila 3
        ("<", 3, 3), ("4", 3, 4),                   # fila 4
        ("<", 4, 0), ("<", 4, 1)                    # fila 5
        ),              # fin de la partida número 1
        ...             # otras partidas
]
```

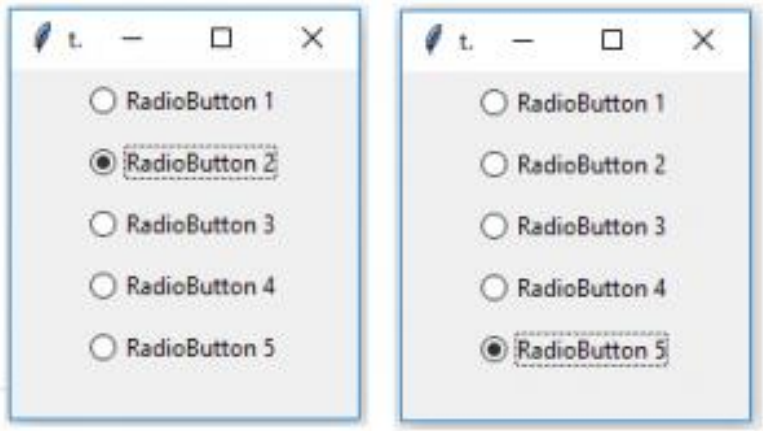
Busque algunas partidas, al menos 3 por cada nivel, y grábelas directamente en el archivo.

Temas Investigados

Radio buttons

Los “Radio Buttons” son un widget común de Tkinter que se utiliza para darle uno de muchos valores a una variable. Los “Radio Buttons” pueden tener texto o imágenes e incluso se pueden asociar a funciones o métodos, tal como un botón normal. Lo que distingue un “Radio Button” de un botón normal es que varios botones comparten una misma función o valor que asignarle a una variable, cuando uno de estos botones es seleccionado la función o valor del resto es ignorada y se ejecuta la función de aquel que fue presionado o se le da ese valor a una variable.

Un ejemplo de radiobutton es:



Como se puede ver en la imagen al seleccionar un botón se va a deseleccionar el resto asignado al mismo valor.

Estos botones también pueden tener la apariencia de un botón normal, de esta forma se verán así:

El comando para llamar uno de estos botones es:

Radiobutton(master, text = ‘texto deseado’, variable = v (Variable a la que se le asignará un valor), value = valor que le dará a la variable)

El texto y aspectos de apariencia del botón también se pueden modificar al igual que con cualquier otro widget.

Función After

after() es una función universal de Tkinter que puede ser utilizada en la ventana principal o con otros widgets. Esta función tiene el propósito de ejecutar un comando o acción dentro de una ventana tras cierta cantidad de microsegundos. Los parámetros que recibe esta función son:

- ➔ after(ventana, cantidad_microsegundos, función a ejecutar)

Generalmente el método after se utiliza para retrasar un proceso para que suceda hasta que pase una cantidad de tiempo, otro uso muy común que tiene es para relojes o timers, para que se cuente cada cierta cantidad de tiempo.

Un ejemplo de after para que realice una acción tras un segundo

- ➔ after(Main, 1000, contar)

Importancia del Software de control de Versiones

También conocido como control de Fuente, el control de versiones se encarga de manejar diversos momentos de un mismo archivo, código, etc. El control de versiones incluye el software de control de versiones. El uso de este software para programación se conoce como VCS programming. El software de control de versiones le permite a múltiples desarrolladores, diseñadores y miembros de un equipo de trabajar en un mismo proyecto e incluso en distintas versiones de este. Estos sistemas son esenciales para asegurarse de que todos tengan la última versión de un código, archivo, etc. Mientras más complejo sea un proyecto, más versiones es necesario controlar, aumentando la utilidad del repertorio con cada versión nueva.

Este tipo de software, como se menciono anteriormente, es importante para mantener un control de los cambios realizados en un trabajo y para mantener a cada persona que esta participando en un proyecto actualizada en cuanto a su estado. Por esta razón si se desea trabajar de forma grupal en un proyecto o si se desea tener acceso a versiones pasadas en caso de que pase algo en la última es esencial su uso.

Usar esta clase de software no solo permite manejar varias versiones de un archivo, también le permite a las persones desarrollar y entregar productos mucho más rápido, ya que, si se comete algún error, su corrección es mucho más veloz y eficiente. Entre los softwares de control de versiones más importantes están:

- ➔ Helix Core (Perforce) , Git, SVN, ClearCase, Mercurial y TFS.

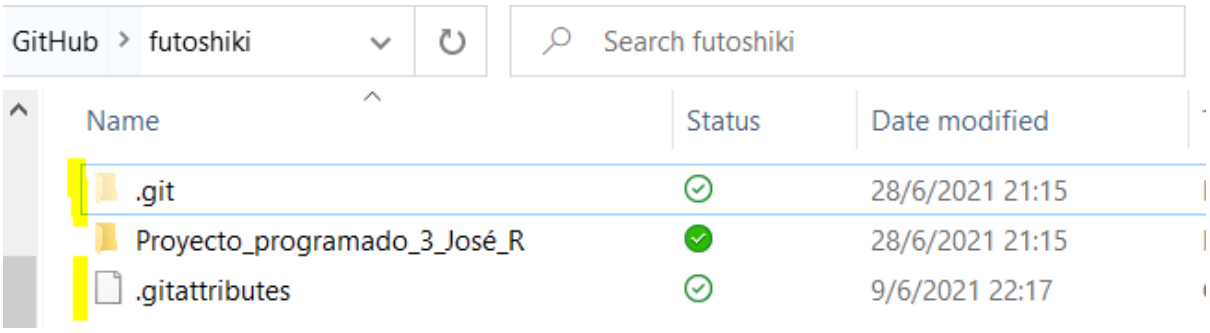
Git (Software de control de versiones usado)

Git es un software de control de versiones de fuente abierta, es decir, cualquiera puede utilizarlo para acceder a los proyectos de otras personas siempre y cuando no sean privados. Hay dos formas de utilizar git una es enviándole comandos en la sección web de GitHub. La otra forma es a través de la aplicación de GitHub.

Para este proyecto la forma que fue utilizada para el desarrollo del proyecto es la aplicación. Por lo tanto, esa es la que se va a explicar.

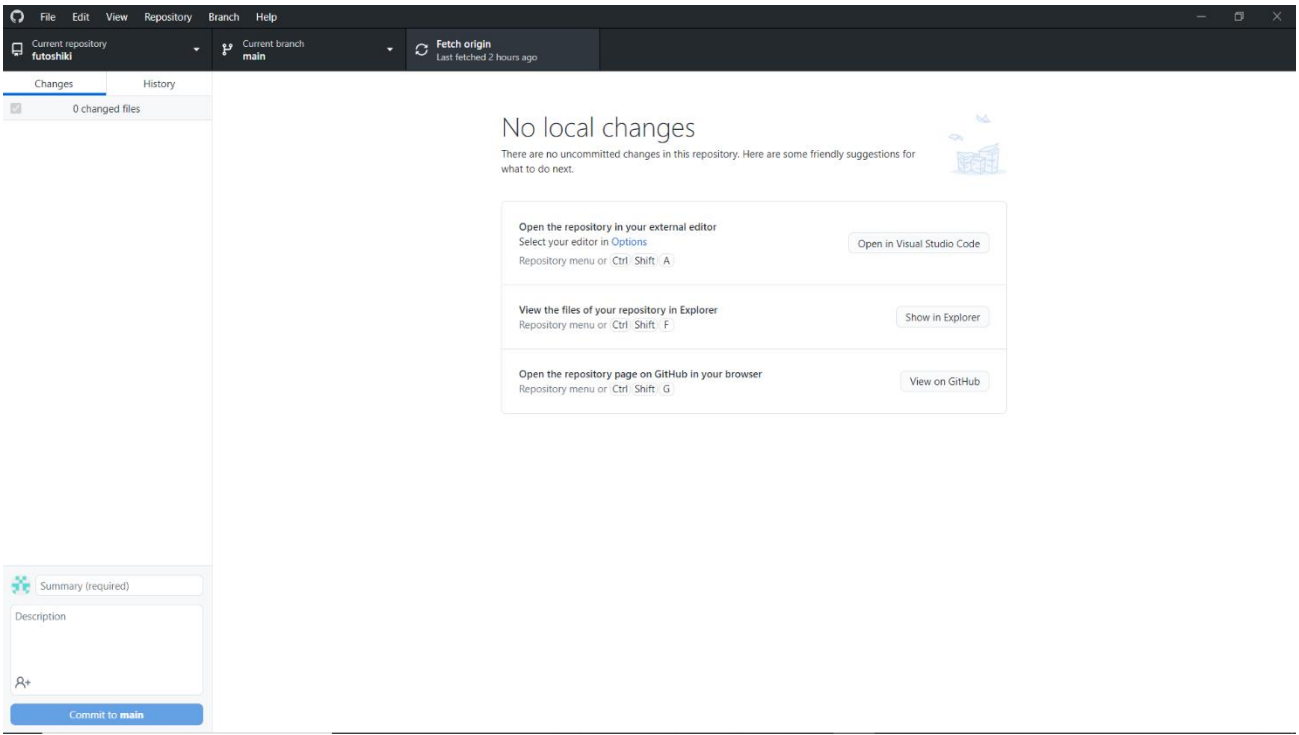
La aplicación de GitHub permite establecer una conexión entre la cuenta en línea de Git junto con sus repertorios a través de una interfaz sencilla y fácil de utilizar. Para instalar la aplicación simplemente es cuestión de entrar a la página de GitHub y descargar el instalador de la aplicación, desde ese punto es cuestión de seguir los pasos indicados por el instalador. Al instalarse, la aplicación de GitHub crea un repertorio bajo el mismo nombre donde se pueden crear y guardar distintos proyectos. Este será el logo del app →

Dentro del repertorio github se debe verificar que estén los archivos siguientes:

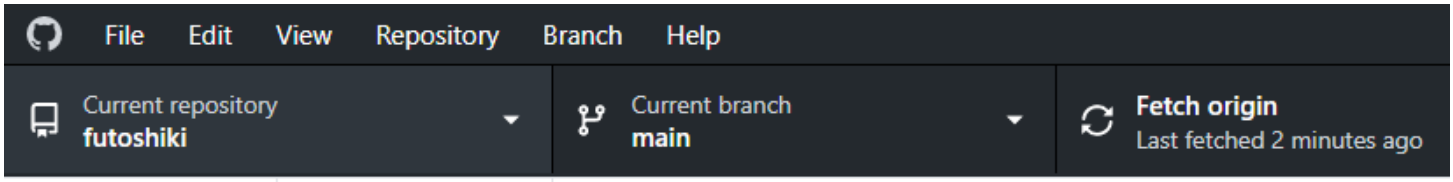


Estos dos archivos indican que se está manejando un repertorio de GitHub.

Para manejar específicamente la aplicación se va a encontrar la siguiente interfaz:



Las opciones del menú barra son:

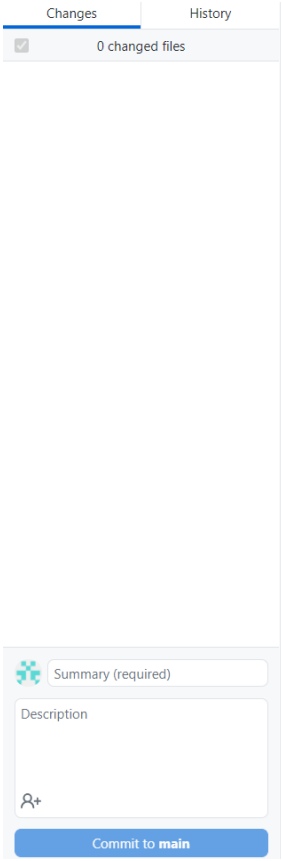


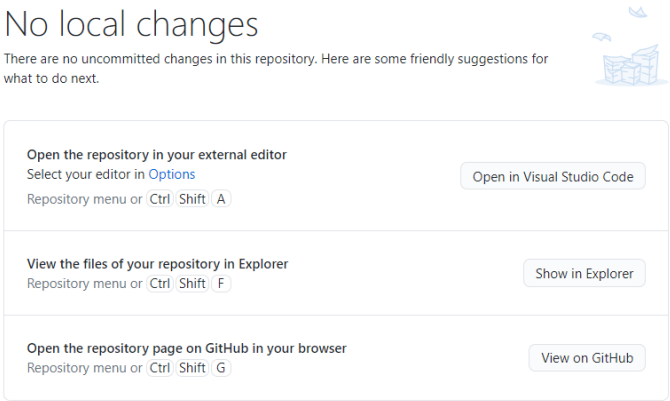
La opción de file permite al usuario crear, clonar y borrar repertorios de GitHub, borrar o crear un nuevo repertorio en la aplicación también creará o borrará un repertorio en la página web de GitHub.

La opción de edit permite realizar undo y redo a acciones que ya se hayan realizado dentro de la aplicación, ya sea que se agrego un archivo, se creo un nuevo repertorio o se editó algún archivo.

La opción fetch origin permite revisar si sucedió algún cambio nuevo en los archivos del repertorio, si ese es el caso se le enviará un mensaje de aviso al usuario.

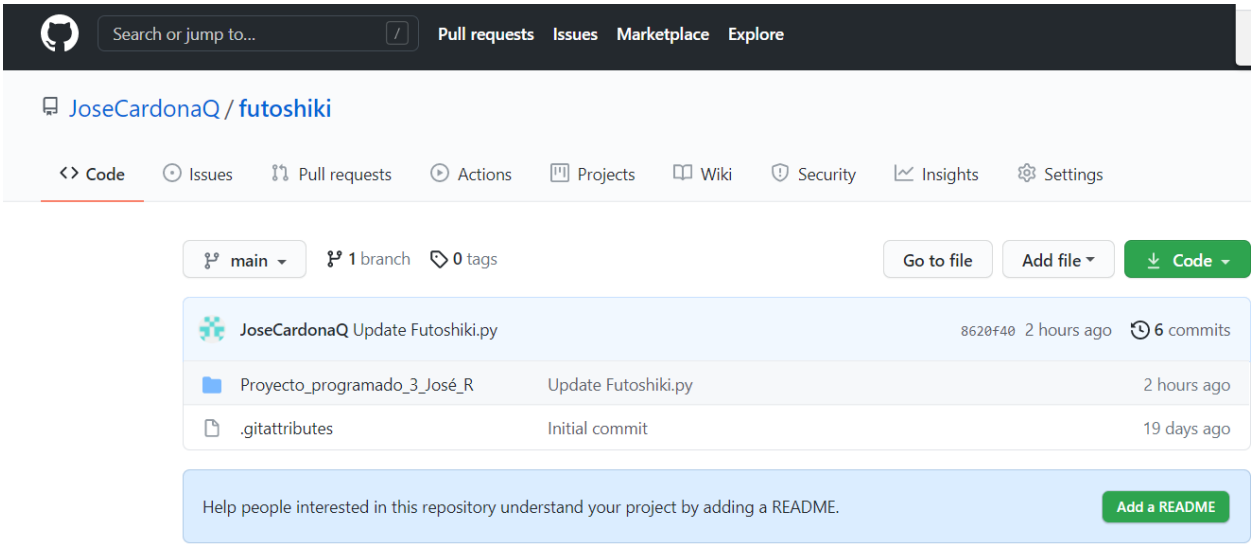
Las opciones dentro de la ventana por otro lado son:



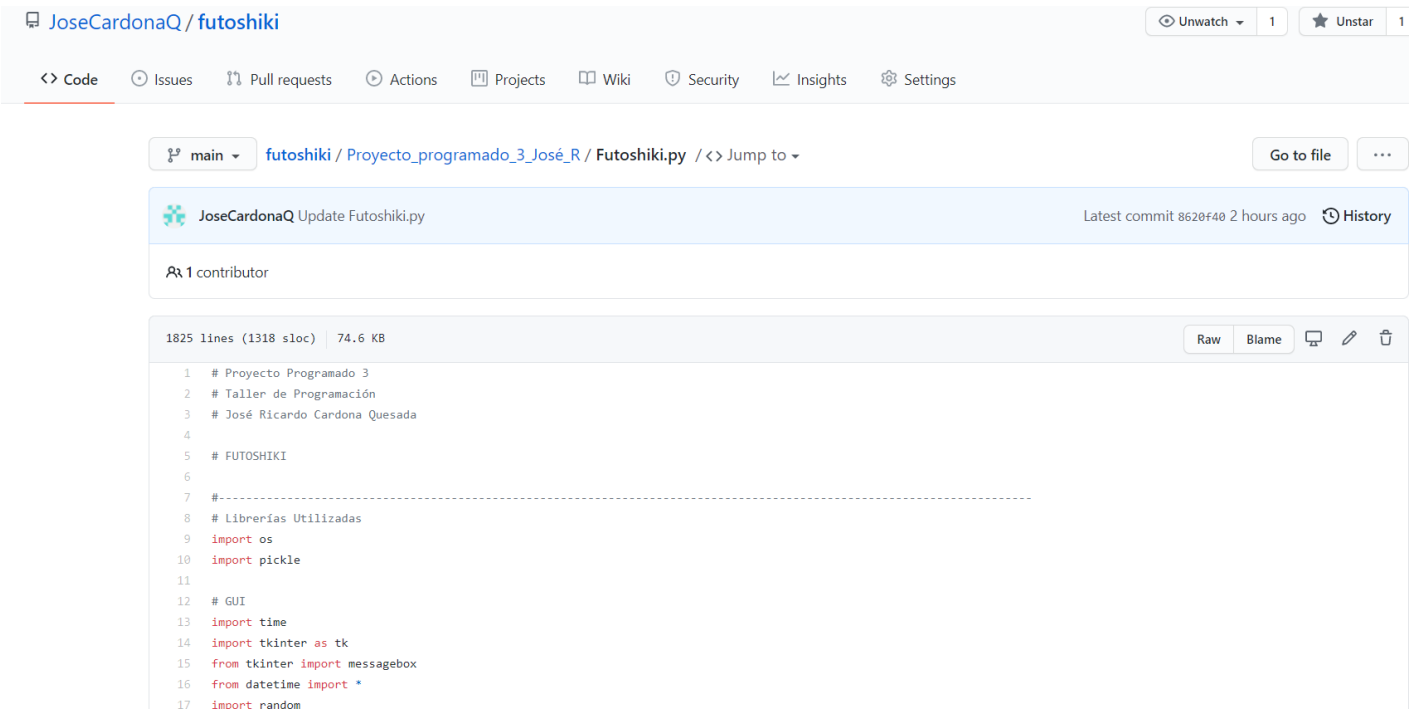


Las opciones a la derecha permiten abrir el repertorio en distintos lugares, la primera opción permite abrir el repertorio en un editor externo, en este caso el editor default es Visual Studio Code. La segunda opción permite abrir el archivo que contiene el repertorio dentro de la computadora. Finalmente, la tercera opción abre el repertorio en la página de GitHub. Este repertorio representará la última versión que fue subida desde la aplicación.

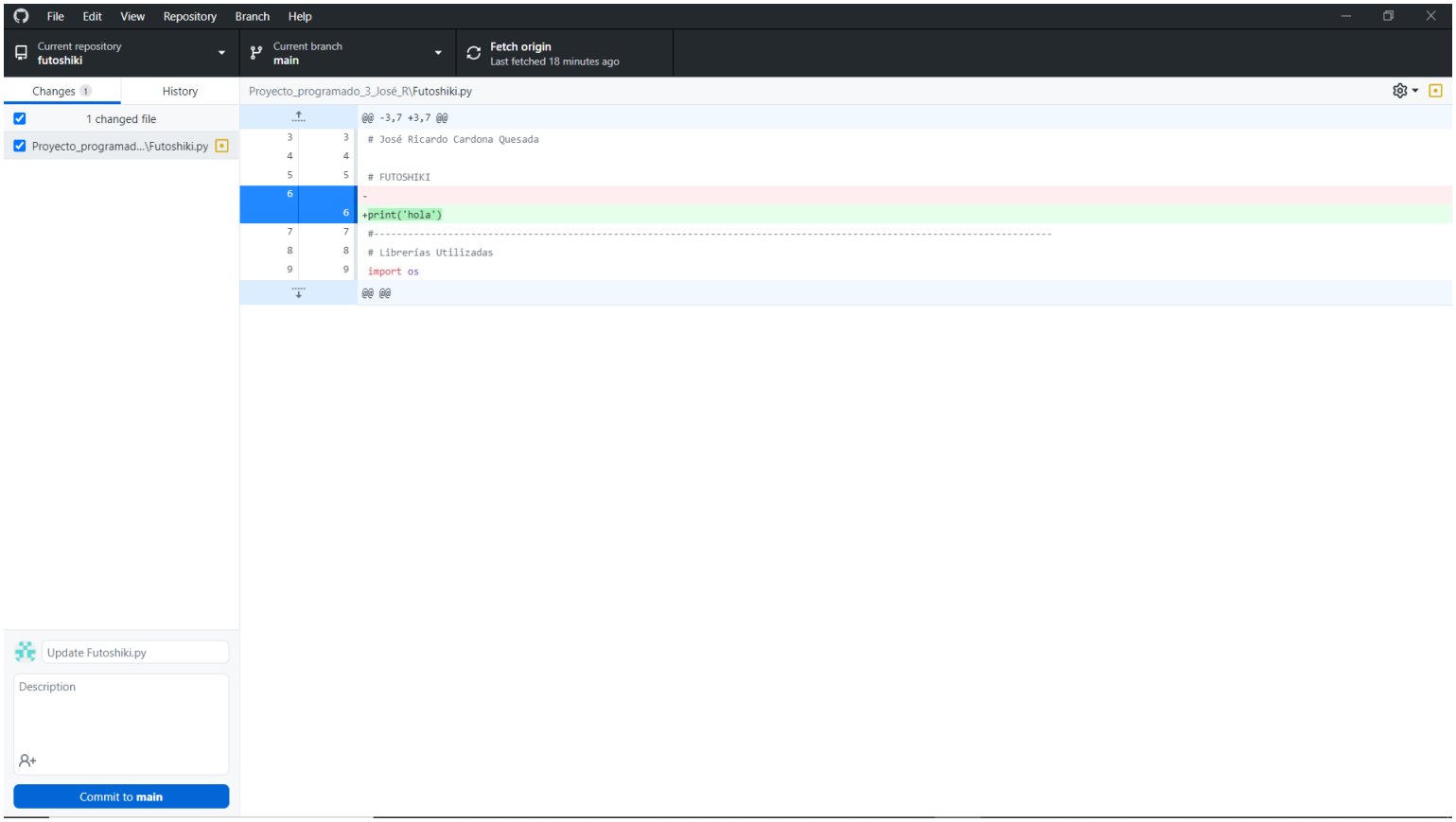
En línea el repertorio se verá de esta forma:



Al igual que una carpeta normal dentro de una computadora, se podrán abrir los archivos dentro del repertorio en línea e incluso se pueden inspeccionar a profundidad. Por ejemplo, si se abre un código en el repertorio en línea de GitHub se ve así:



Para concluir, es importante remarcar que al cambiar un archivo dentro del repertorio local no se actualiza automáticamente dentro del repertorio en línea. Para actualizarlo es necesario abrir la aplicación tras haber hecho el cambio local, al hacer eso la aplicación desplegará el mensaje:



Como se ve en la imagen, cuando se abre la aplicación tras el cambio se desplegará el archivo y los cambios realizados, en caso de que se quiera guardar el cambio se presiona el botón con el texto: “Commit to Main”, hacer esto guardará el cambio como oficial en el repertorio en línea de GitHub.

Conclusiones

- Anteriormente no podía entender como es que funcionaban los relojes y los timers dentro de una aplicación, sin embargo, gracias a este proyecto, aprendí que el método utilizado es el de after() o al menos uno similar que permita ejecutar una función hasta que suceda cierta cantidad de tiempo.

- Previo a este trabajo el método que había utilizado para controlar las versiones de mis archivos era simplemente hacer una copia, sin embargo, gracias a este trabajo y la herramienta GitHub aprendí un método mucho más eficiente y fácil de tener un control sobre las versiones de todo lo que hago. Gracias a esto no solo me voy a volver una persona más eficiente en cuanto a proyectos y trabajos, sino que también más ordenado.

- A pesar de que el uso de GitHub fue limitado en el proyecto, quedo claro que este junto con otros softwares de control de versiones son herramientas extremadamente necesarias para cualquier programador que quiera trabajar de forma profesional en la carrera de computación y que quiera tener éxito con todos sus proyectos.

- A pesar de su aspecto un poco anticuado, gracias al trabajo en este proyecto, pude observar que Tkinter es increíblemente útil y se puede adaptar a una variedad de situaciones, ya sea un menú de administración de parqueo o un pasatiempo como el Futoshiki.

- A la hora de presentarle al usuario una selección de opciones donde solo se pueda seleccionar una, la mejor opción es utilizar alguna forma de Radio Button, aunque tenga un nombre diferente, mientras sea algo que permita realizar el mismo propósito

- El Futoshiki no es un pasatiempo sencillo, aunque así lo parezca, y el proceso de programación de este juego dentro de Python tampoco lo es. Al inicio mis pensamientos eran que iba a ser sumamente fácil la programación del programa, sin embargo, realmente me sorprendió la complejidad tras la creación de un pasatiempo tan sencillo. Esto simplemente me hizo apreciar aún más los esfuerzos de todos los programadores de juegos y el trabajo que hay tras un juego.

Diseño y Explicación de la Solución

Configuración

Para los valores de configuración se utiliza el archivo llamado “futoshiki2021configuración”, en este existe una lista que contiene cada uno de los datos de la configuración en este orden:

[dificultad , config_reloj, posición_panel, horas_timer, minutos_timer, segundos_timer]

Cada uno de estos valores se extrae a través del método pickle y se le van asignando los valores respectivos a variables de Tkinter. Dentro del programa esto se ve de la siguiente manera →

```
config = pickle.load(configuracion)

# La Dificultad
# Tiene 3 posibilidades: Facil, Intermedio, Dificil
# Comienza como Fácil la primera vez que se corra el programa
dificultad = tk.StringVar()
dificultad.set(config[0])

# El Reloj
# Puede ser Si, No o Timer
# Comienza como Si la primera vez que se corra el programa
reloj = tk.StringVar()
reloj.set(config[1])

# La posición en la ventana
# Puede ser Derecha o Izquierda
# La primera vez comienza como derecha
position = tk.StringVar()
position.set(config[2])

# Valores para el timer
# Solo se van a utilizar cuando el reloj sea timer
horas_timer = tk.IntVar()
horas_timer.set(config[3])

# A diferencia de el resto esta  variable comienza como 30
minutos_timer = tk.IntVar()
minutos_timer.set(config[4])

segundos_timer = tk.IntVar()
segundos_timer.set(config[5])

configuracion.close()
```

Posteriormente estas variables son asignadas a los diversos radiobuttons dentro del menú de configuración y dependiendo del valor que tenga asignado el radiobutton ese valor se le dará a la variable.

Estos valores se guardan dentro del documento DAT con la lista en el momento que el usuario cierra la ventana principal del programa, esto se realiza a través del método pickle.dump()

Jugar

Seleccionar Partida

Para la opción de jugar al Futoshiki se debe de seleccionar uno de los mapas para la dificultad asignada, sin embargo, la selección de este mapa dependerá de si se está cargando una partida o se está intentando jugar una partida nueva. Al presionar el botón Jugar siempre se asume que se esta jugando una partida nueva y por lo tanto la entrada a cargar siempre será una lista vacía, en el punto que esta entrada no sea una lista vacía se asume que el programa esta cargando una partida y por ende cargará el mapa que se le dio de entrada.

Las partidas de Futoshiki están guardadas dentro del archivo con el nombre de "futoshiki2021partidas" y este contiene una lista que al mismo tiempo contiene 3 listas. Cada una de estas listas representa una dificultad. Por dificultad están guardadas 3 partidas, cada una de estas partidas contenida dentro de una lista propia. Básicamente se vería así:

[[partida_1, partida_2, partida_3] , [partida_1, partida_2, partida_3], [partida_1, partida_2, partida_3]]

Fácil Intermedio Difícil

Para la selección de partidas en caso de partida nueva se utilizaron dos cosas, la primera es el valor que tenga asignado la variable de Tkinter "dificultad" este determina cual índice de la lista de partidas se quiere acceder. Luego de obtener el índice de dificultad se necesita seleccionar una partida. Para esto se utilizó el método:

Randint() el cual permite obtener un número aleatorio de un rango de números dados, ya que cada dificultad solo tiene 3 partidas el rango estará entre 0 y 2. Este número obtenido será el índice de la partida que se va a jugar. En el código esto se verá así:

```
def jugar_futoshiki(cargado):

    # Según el nivel de dificultad se importará aleatoriamente
    # Para seleccionar aleatoriamente se va a utilizar el métod

    if cargado == []:

        if dificultad.get() == 'Facil':
            mapa = partidas_faciles[random.randint(0, 2)]

        elif dificultad.get() == 'Intermedio':
            mapa = partidas_regulares[random.randint(0, 2)]

        else:
            mapa = partidas_dificiles[random.randint(0, 2)]

    else:
        mapa = cargado[1]
```

Desplegar una partida

Como fue indicado en la descripción del proyecto cada partida tiene la forma:

```
[           # lista de partidas de nivel fácil:
        (               # tupla de la partida número 1
        (">", 0, 0), (">", 0, 2), (">", 0, 3),      # fila 1
        ("4", 1, 0), ("2", 1, 4),                   # fila 2
        ("4", 2, 2),                                # fila 3
        ("<", 3, 3), ("4", 3, 4),                   # fila 4
        ("<", 4, 0), ("<", 4, 1)                    # fila 5
        ),              # fin de la partida número 1
        ...             # otras partidas
]
```

Como se ve en la imagen cada partida indica valores e indica las filas y columnas a las que pertenecen estos valores. Esto indica que la forma en que se deben tratar los botones es como si fueran una matriz cuadrada de 5x5. Para esto primero se crean los 25 botones que siempre existirán y dependiendo de las opciones de configuración se acomodan el resto de los objetos como el panel de dígitos, y el reloj. Todos los botones se contendrán dentro de una lista bajo el nombre "Tiles". Esta lista se verá así:

```
# Se guardan todos los botones en una lista, esta contendrá cad
tiles = [[lab_0_0,lab_0_1,lab_0_2,lab_0_3,lab_0_4], # Fila 1
         [lab_1_0,lab_1_1,lab_1_2,lab_1_3,lab_1_4], # Fila 2
         [lab_2_0,lab_2_1,lab_2_2,lab_2_3,lab_2_4], # Fila 3
         [lab_3_0,lab_3_1,lab_3_2,lab_3_3,lab_3_4], # Fila 4
         [lab_4_0,lab_4_1,lab_4_2,lab_4_3,lab_4_4]] # Fila 5
```

Luego de acomodar todos botones y elementos se procede a incluir todas las restricciones horizontales, verticales. Para esto también se forman las respectivas filas y columnas, sin embargo, están restricciones serán labels en lugar de botones. Las restricciones horizontales solo tienen 4 columnas y las restricciones verticales solo tienen 4 filas. Las respectivas listas para ambas restricciones son:

```
# Se guardan estos labels en una lista
restricciones_horizontales = [[hor_restr_0_0,hor_restr_0_1,hor_restr_0_2,hor_restr_0_3],
                              [hor_restr_1_0,hor_restr_1_1,hor_restr_1_2,hor_restr_1_3],
                              [hor_restr_2_0,hor_restr_2_1,hor_restr_2_2,hor_restr_2_3],
                              [hor_restr_3_0,hor_restr_3_1,hor_restr_3_2,hor_restr_3_3],
                              [hor_restr_4_0,hor_restr_4_1,hor_restr_4_2,hor_restr_4_3]]
```

```
restricciones_verticales = [[ver_restr_0_0,ver_restr_0_1,ver_restr_0_2,ver_restr_0_3,ver_restr_0_4],
                            [ver_restr_1_0,ver_restr_1_1,ver_restr_1_2,ver_restr_1_3,ver_restr_1_4],
                            [ver_restr_2_0,ver_restr_2_1,ver_restr_2_2,ver_restr_2_3,ver_restr_2_4],
                            [ver_restr_3_0,ver_restr_3_1,ver_restr_3_2,ver_restr_3_3,ver_restr_3_4]]
```

Inicialmente todos estos labels de restricción comienzan como espacios vacíos. Sin embargo, posteriormente se utiliza un ciclo de tipo for para navegar las restricciones y si se encuentra una restricción de ese tipo se le asigna a la fila y columna de la restricción, cambiando de esta forma su texto por la restricción. Esto se verá de la siguiente manera:

```
# Por cada restricción, si se encuentra una comparación horizontal se cambia el indice al que pertenece
for restriccion in mapa:
    if restriccion[0] == ">" or restriccion[0] == "<":
        restricciones_horizontales[restriccion[1]][restriccion[2]].config(text = restriccion[0])
```

```
# Por cada restricción, si se encuentra una comparación vertical se cambia el indice al que pertenece
for restriccion in mapa:
    if restriccion[0] == '∧' or restriccion[0] == 'V':
        restricciones_verticales[restriccion[1]][restriccion[2]].config(text=restriccion[0])
```

Hacer esto permite que se desplieguen únicamente aquellos labels que si presentan una restricción completando de esta forma el mapa. Finalmente, para los valores fijos se utiliza otro ciclo de for para asignarle a aquellos botones que lo requieran un valor

```
# Para los números fijos
for restriccion in mapa:
    if restriccion[0].isdigit():
        tiles[restriccion[1]][restriccion[2]].config(text = restriccion[0])
```

Posteriormente a la hora de seleccionar el botón de iniciar juego se corre el mismo ciclo, sin embargo, en lugar de asignar el valor nuevamente, se deshabilitan los botones con una restricción de este tipo.

Darle un valor a un botón

Para darle un valor a un botón se utiliza una función que recibe el botón al que se le quiere asignar el valor y se empieza a verificar si cumple con las diversas restricciones que tiene. Se verifica que el valor que se le quiere dar no exista en la misma fila o columna de la matriz de botones en caso de que esto se cumpla se verifica que de acuerdo con las localizaciones de las restricciones verticales y horizontales que exista el botón cumpla aquellas restricciones en las filas y columnas a su alrededor. Para todas estas restricciones se utilizan ciclos y la localización de fila y columna del botón. En caso de que todas las restricciones se cumplan se configura el texto del botón dado por el valor del panel de dígitos que se le quería asignar. Esto se realizo a través del método .**config()**

Iniciar Juego

Iniciar juego es una simple función que obtiene el valor dentro del entry de nombre y recorre a través de ciclos la lista de top 10, si no encuentra el nombre ni lo registra como vacío va a habilitar el resto de los botones que habían estado deshabilitados hasta ese momento para que el usuario pueda jugar. Simultáneamente este botón va a iniciar la función recursiva llamada reloj_timer() para que se comience a contar. Finalmente Iniciar juego le asigna un tiempo inicial al juego para poder contar el tiempo total que dure el usuario realizando la partida y compararlo posteriormente en el top10. Este tiempo se le asigna a una variable bajo el nombre de tiempo_inicial.

Borrar Jugada

Borrar Jugada Funciona a través del uso de una lista, esta tiene el nombre de “lista_jugadas”. Si se empezó una partida nueva la lista comienza como vacía y con cada jugada correcta que se hace se le agrega un elemento. Los elementos que se agregarán a esta serán tuplas y contendrán el valor que tenía el botón antes de cambiar junto con la fila y columna donde esta ubicado el botón. Esto se verá así:

(Valor pasado botón , fila, columna)

Un ejemplo de una lista de jugadas con 3 jugadas hechas sería:

lista_jugadas = [(“ ”,3,0) , (“2”,3,0) , (“ ”, 1, 0)]

La forma en la que funciona el botón borrar jugada es que sigue la metodología LIFO, es decir el último valor en entrar es el primero en salir. Por esto cuando se presiona el botón borrar jugada se saca el índice -1 (Último de la lista) y se le retorna al botón en esa fila y columna el valor guardado dentro de la tupla. Para esto se utiliza el método de manejo de secuencias **pop().**

En el programa esto se ve así:

```
# Se saca el último valor que entro a la lista LIFO
jugada_pasada = lista_jugadas.pop(-1)

# Se retorna ese valor pasado a el botón al que pertenecía
tiles[jugada_pasada[1]][jugada_pasada[2]].config(text = jugada_pasada[0])
```

En caso de no tener más jugadas en la lista se retorna un Messagebox con el mensaje respectivo.

Terminar Juego y Borrar Juego

Ambas de estas son funciones que utilizan una versión de el objeto de tipo Messagebox, esta versión será Messagebox.askquestion y presentan opciones de si y no al usuario. En ambas funciones al presionar el botón no se cancela toda acción y la función no hace nada. Al marcar que si en la función Terminar juego se cierra la ventana utilizando el método destroy() y se abre una ventana nueva corriendo la función jugar_futoshiki([]) nuevamente. La función borrar jugada, por otro lado, vacía la lista de jugadas, la cual se llama **“lista_jugadas”** y borra todos los valores de los botones que no sean fijos.

Guardar y Cargar Juego

Ambas funciones hacen uso del archivo bajo el nombre de " futoshiki2021juegoactual" este documento será una lista que contiene todos los datos de la partida que haya sido guardada por el usuario.

Está lista contendrá:

[

Lista de jugadas,

Mapa utilizado,

(dificultad , nombre del jugador , ('horas reloj' , 'minutos reloj' , 'segundos reloj'),

Valores_botones

]

La lista de jugadas será la misma lista que se utiliza dentro de la función para borrar jugadas, el mapa es aquel que se utilizo para formar la partida guardada, la tupla con valores contiene diversos datos de la partida pasada. Finalmente contendrá una lista con los valores de cada uno de los botones, esta lista de valores se forma previó a guardar el documento y tendrá la misma cantidad de filas y columnas que la lista "Tiles". La diferencia entre la lista Tiles y esta es el contenido de cada columna, ya que no se pueden guardar objetos como botones, es necesario guardar el valor de cada uno de los botones de forma diferente. Por ende, cada columna en la lista de valores contendrá una tupla con el valor del botón respectivo a esa columna y la fila y columna donde estaba ese botón.

Durante la función de guardar partida se forma esta lista de la siguiente manera →

```
# Se forma una lista con los valores actuales de cada tile
valores = []

for fila in range(len(tiles)):
    f = []
    for columna in range(len(tiles[fila])):

        f.append((tiles[fila][columna]['text'], fila,columna))

    valores.append(f)
```

Posterior a la creación de la lista se utiliza el método pickle.dump() para guardar en el documento de partida actual los datos obtenidos de la partida que se decidió guardar.

```
# Se crea una lista con los datos necesarios
datos_g = [lista_jugadas,mapa,(dificultad.get(),nombre_jugador.get(),(horas.get(),minutos.get(),segundos.get()))
           ,valores]

# Se guarda la lista en el archivo
pickle.dump(datos_g,guardar_j)
guardar_j.close()
```

Cargar Juego , por otro lado, no guarda un valor nuevo dentro del documento de partida actual, en lugar de eso, el botón cierra la ventana actual de Futoshiki y abre una nueva, sin embargo, al hacer esto lo hace dándole de entrada la lista con todos los valores de la partida guardada. Esto lo realiza a través de el método pickle.load() y asignándole esa lista a la entrada de la función jugar_futoshiki().\

```
# Función para cargar el juego
def cargar_juego():

    # Se abre el archivo de juego actual
    juego_c = open('futoshiki2021juegoactual.dat','rb')
    juego_cargado = pickle.load(juego_c)

    # Se destruye la ventana actual
    futoshiki.destroy()

    # Se abre una ventana nueva con el mapa nuevo y los datos obtenidos
    jugar_futoshiki(juego_cargado)
```

En base a esto muchos de los valores de la ventana de jugar Futoshiki que aparecían como nuevos obtienen los valores de la partida pasada, entre estos los valores de los tiles que ya tuvieran valores asignados la partida pasada, los tiempos en el timer / reloj, el nombre del jugador, la lista de jugadas, etc.

Top10

Finalmente, el top 10 esta contenido dentro del archivo bajo el nombre de “futoshiki2021top10 “ y consiste en una lista que al mismo tiempo contiene 3 listas. Cada una de estas listas contiene la información de los usuarios que pertenecen al top 10 en una dificultad. El máximo de usuarios por lista es de 10 y cada usuario es representado por una tupla que contiene dos valores el nombre del usuario y el tiempo que le tomo completar el Futoshiki. Un ejemplo de una lista de top 10 sería:

[[(“Marco” , ’01:05:15’), (‘María’ , ‘02 : 02 : 19’)] , [] , [(“Robert” , ’00:59:15’), (“Flora” , ’01:41:15’)]

Fácil Intermedio Difícil

Esta lista se abre a través del método pickle y se guarda de la misma forma al cerrar el programa. La opción top 10 dentro de jugar utiliza ciclos para crear un Label con la información de cada jugador por dificultad en la lista de top 10.

Estadística de tiempos

Ventana Principal	Horas
Botones , Imágenes, etc..	1 / 2
Salir	1 / 2
Guardar Archivos	1
Cargar Archivos	1
Total	**3**

Configuración	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	1 / 2
Investigación	1
Programación	1
Documentación interna	1 / 2
Pruebas	1
Elaboración del manual de usuario	1 / 2
Elaboración de documentación del proyecto	1
Total	**6**

Despliegue Mapa	Horas
Análisis del problema	1
Diseño de algoritmos	2
Investigación	0
Programación	1
Documentación interna	1 / 2
Pruebas	2
Total	**6, 5**

Reloj / Timer	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	1
Investigación	1
Programación	2
Documentación interna	1 / 2
Pruebas	1
Total	**6**

Iniciar Juego	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	1 / 2
Investigación	0
Programación	1 / 2
Documentación interna	1 / 2
Pruebas	1 / 2
Elaboración del manual de usuario	1 / 2
Elaboración de documentación del proyecto	1
Total	**4**

Activar Tile	Horas
Análisis del problema	2
Restricción Valor Fila	1 / 2
Restricción Valor Columna	1/2
Restricción Horizontal	2
Restricción Vertical	2
Pruebas	4
Total	**11**

Borrar Jugada	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	1 / 2
Investigación	0
Programación	1 / 2
Documentación interna	1 / 2
Pruebas	1 / 2
Elaboración del manual de usuario	1 / 2
Elaboración de documentación del proyecto	1
Total	**4**

Terminar Juego	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	1 / 2
Investigación	1
Programación	1
Documentación interna	1 / 2
Pruebas	1 / 2
Elaboración del manual de usuario	1 / 2
Elaboración de documentación del proyecto	1
Total	**5,5**

Borrar Juego	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	1 / 2
Investigación	0
Programación	1
Documentación interna	1 / 2
Pruebas	1 / 2
Elaboración del manual de usuario	1 / 2
Elaboración de documentación del proyecto	1
Total	**4,5**

Top 10	Horas
Análisis del problema	1
Diseño de algoritmos	1 / 2
Investigación	0
Programación	2
Documentación interna	1 / 2
Pruebas	1
Elaboración del manual de usuario	1
Elaboración de documentación del proyecto	1
Total	**7**

Cargar Juego	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	2
Investigación	0
Programación	1 / 2
Documentación interna	1
Pruebas	2
Elaboración del manual de usuario	1 / 2
Elaboración de documentación del proyecto	1
Arreglos y Correcciones Algoritmo Pasado de Juego	2
Total	**9,5**

Guardar Juego	Horas
Análisis del problema	1 / 2
Diseño de algoritmos	1 / 2
Investigación	0
Programación	1 / 2
Documentación interna	1 / 2
Pruebas	1 / 2
Elaboración del manual de usuario	1 / 2
Elaboración de documentación del proyecto	1
Total	**4**

REVISIÓN DEL PROYECTO

Concepto	Puntos originales	Avance 100/%/0	Puntos obtenidos	Análisis de resultados
Opción Configurar	6	100 %	6	
Despliegue y manipulación de la ventana del juego: cuadrícula con sus restricciones y dígitos fijos. Incluye el despliegue de partidas.	12	100 %	12	
Despliegue y manipulación de la ventana del juego: otros elementos	6	100 %		
Botón Iniciar Juego	10	100 %		
Crear Top 10	12	100 %	12	
Botón Borrar Jugada	5	100 %	5	
Botón Terminar Juego	2	100 %	2	
Botón Borrar Juego	2	100 %	2	
Botón Top 10	10	90 %	9	Despliega el top 10 a la perfección, sin embargo, no detiene el reloj a la hora de abrir el top 10, por lo tanto, el timer o el reloj seguirán contando mientras siga abierta la ventana de top 10.
Botón Guardar Juego	5	100 %	5	
Botón Cargar Juego (incluye el despliegue del mismo)	15	100 %	15	
Ayuda (manual de usuario)	5	100 %	5	
Reloj tiempo real	5	100 %	5	
Timer tiempo real	5	100 %	5	
TOTAL	**100**			

Bibliografía

RadioButton in Tkinter. (2019, mayo 20). Geeksforgeeks.org. https://www.geeksforgeeks.org/radiobutton-in-tkinter-python/

After method in Tkinter. (2019, abril 8). Geeksforgeeks.org. https://www.geeksforgeeks.org/python-after-method-in-tkinter/

after method in Python Tkinter. (s/f). Tutorialspoint.com. Recuperado el 29 de junio de 2021, de https://www.tutorialspoint.com/after-method-in-python-tkinter

tkinter.messagebox — Tkinter message prompts — documentación de Python - 3.9.6. (s/f). Python.org. Recuperado el 29 de junio de 2021, de https://docs.python.org/es/3/library/tkinter.messagebox.html