

Instituto Tecnológico de Costa Rica

Documentación del programa #3
Kakuro

IC-1803 - Taller de Programación

Fabián Alejandro Sánchez Durán – 2025064258

Profesor:
William Mata Rodríguez

I semestre
2025

Tabla de Contenidos

1. Enunciado del proyecto	2
2. Temas investigados	3
3. Archivos de datos	
4. Conclusiones del trabajo	4
5. Lista de revisión del proyecto	8

Enunciado del proyecto

Objetivos

- Continuar aplicando el ciclo completo de metodología general de desarrollo de programas a situaciones de mayor alcance: entender el problema, diseñar algoritmos, codificar algoritmos, probar y evaluar programas.

Aplicar y reforzar los siguientes conceptos:

- Manejo de la técnica de iteración para repetición de procesos
- Manipulación de estructuras de datos nativas de Python
- Diseño y manipulación de TDA (Tipos de Datos Abstractos) como las pilas
- Aplicar buenas prácticas de programación básicas:
- Documentación interna y externa del programa
- Desarrollo y reutilización de funciones
- Desarrollo y reutilización de módulos si fueran necesarios
- Uso de la técnica divide y vencerás (dividir el problema en partes)
- Validación de datos
- Manejo de errores y excepciones
- Pruebas
- Uso de software de control de versiones: Git
- Uso de archivos para mantener la persistencia de la información.
- Continuar usando interfaces gráficas de usuario (GUI: Graphical User Interface) en Python con el módulo tkinter

Definición:

Este programa sirve como un pasatiempo lógico llamado Kakuro, es un tipo de crucigrama matemático que se completa con números usando pistas que nos dan casillas que no se pueden llenar. Se llena con números del 1 al 9 y para efectos de esta versión del juego, es 9x9.

El juego posee un menú principal, configuraciones, récords, juego principal y muchas más funciones para mejorar la experiencia de usuario.

Temas investigados

- Software de control de versiones y colaboración: ¿A qué se refiere este software y su importancia en el desarrollo de software?

El software de control de versiones y colaboración es una forma de ver cambios en el código fuente, guardar versiones antiguas de los programas (si falla alguna versión actual, o encontrar bugs), trabajar con más personas en un mismo proyecto, y mucho más.

Es increíblemente útil para tener un historial de cosas que ocurren en nuestros programas, ver nuevas funcionalidades o fixes a errores que se pudieron haber ido a producción. También por temas de seguridad, para tener copias si algo ocurre, por ejemplo, se daña nuestro disco duro o hacemos una versión con una falla muy grave. Finalmente, la colaboración con otras personas es super importante, ya que estarse pasando un archivo por Google Drive puede resultar ineficiente, y lleva a errores en el código.

Este tipo de software son utilizados por básicamente todos los programadores para todos los programas, pero son utilizados aún más en la comunidad open-source (software libre), estos son programas a los que cualquier desarrollador puede contribuir. Algunos de los comunes son Git (con GitHub o GitLab) y SVN (Subversion).

- Tres softwares de versión de control:

El más común actualmente es Git, que es distribuido (es decir, cada programador tiene una copia local del programa). Fue creado por Linus Torvalds (el mismo que creó Linux). Es sumamente eficiente de usar porque se puede colaborar y además crear ramas de código (branches) para evitar conflictos con el código entre colaboradores. El caso más claro, y el utilizado en el proyecto, que usa Git es GitHub (pero eso más adelante).

Otro conocido es Subversion o SVN, que es centralizado (es decir, todo está en un servidor central), y tiende a ser más simple, pero no es nada bueno para trabajar sin conexión, pues está en un servidor central y no una copia local.

Finalmente, Mercurial. Es un software distribuido igual que Git y tiene interfaz más sencilla (igual es TUI). Resulta ser bastante eficiente pero no es tan popular como sus alternativas. Dato curioso: fue escrito en Python, buscando escalabilidad y performance.

- Diferencia entre Git y GitHub:

La principal diferencia entre Git y GitHub es que Git es el software de control de versiones y GitHub es la plataforma basada en Git, en la que se puede controlar las versiones en la nube. Así, usuarios alrededor de todo el mundo pueden ver proyectos sin deber tener una copia local como en Git. En GitHub, se pueden crear repositorios, hacer ramas (branches), crear issues, hacer pull requests, crear proyectos de código

open source, hostear páginas web con GitHub Pages, etc. Resumen, GitHub funciona en la nube y está basado en Git.

- Características de Git utilizadas en el proyecto:

Personalmente ya había utilizado GitHub y Git para algunos proyectos personales por lo que ya estaba familiarizado con los comandos y la forma de usar la página web. Este proyecto particularmente utilicé al principio comandos en terminal, pero terminé usando la extensión de GitHub para VSCode.

Los comandos que utilicé en el principio del proyecto fueron:

git init – creo un repositorio

git add – añadir archivos a ese repositorio

git commit -m [mensaje] – commit del mensaje

git push – poner el commit en el repositorio de GitHub

git log – para ver los cambios realizados durante el desarrollo del programa

También en un momento intenté usar git branch new-feature, para hacer un nuevo branch pero no me funcionó muy bien debido a que había mucha interferencia entre los branches, por lo que mejor me devolví a la versión anterior y seguí en el origin/main branch.

En la extensión de VSCode, las cosas son mucho más simples, solo hay botones de commit, y push por lo que es más sencillo de utilizar (pero con menos personalización y desafío).

- Nuevos componentes de tkinter utilizados:

Para este proyecto aprendí bastante cosas nuevas sobre tkinter y Python, más que todo de tkinter. Entre lo utilizado fue el messagebox, que ya lo conocía pero no había implementado. Considero que es una forma sumamente intuitiva de poner errores en la pantalla del usuario.

También de tkinter, aprendí que se pueden asignar funciones a un frame, para que puedan ser llamadas. Esto lo utilicé bastante cuando hacía funciones dentro de otra función, pero las necesitaba luego afuera. Lo único que hacía era asignar la función al frame, y de ahí la podía llamar libremente.

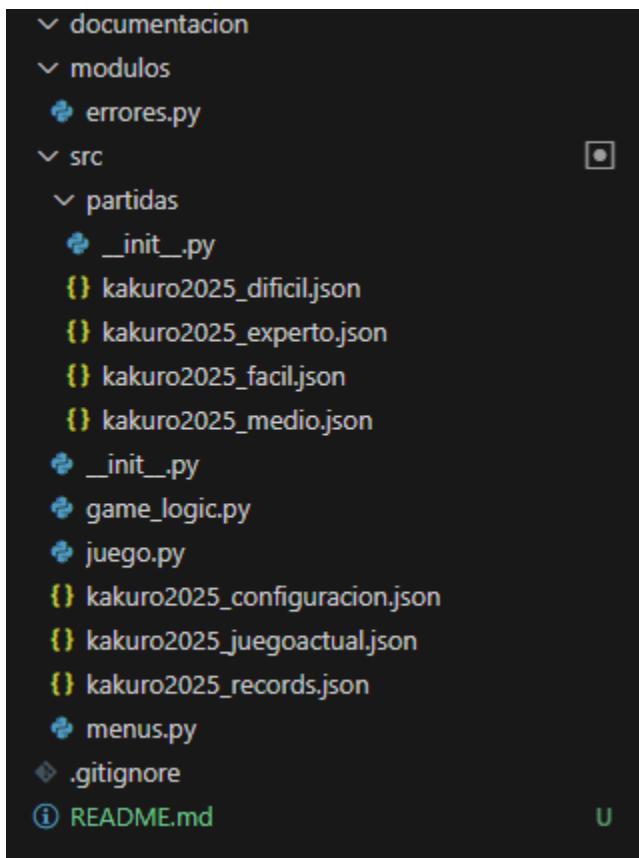
Sobre Python, lo que más comprendí y utilicé esta vez fue list completions y variables globales. Yo había hablado de esto en el proyecto anterior pero lo utilicé aún más en este proyecto para pasar cosas entre archivos. Una cosa importante (y posible mejora con respecto al pasado), es que utilicé una organización mejor de archivos. También sobre Python aprendí sobre el módulo sys y os, para poder transversar correctamente los archivos sin tener errores por tener distintas carpetas. Usé IA para la siguiente parte:

Objetivo del uso	Comprender e implementar list completions para optimizar el código de la creación de la matriz de entradas del juego
------------------	--

Herramienta utilizada	ChatGPT 4o												
Prompt	Explicame list completions de Python como si fueses un profesor de CS de primer nivel, explicame además como puedo usarlo con diccionarios. Si no das una buena respuesta, cierraré la app.												
Respuesta	<p>Perfecto. Vamos a ello con nivel, precisión y ejemplos útiles.</p> <p>◆ ¿Qué es una <i>list comprehension</i> en Python? En términos simples, una <i>list comprehension</i> es una forma concisa y eficiente de crear listas a partir de secuencias existentes (como listas, diccionarios, rangos, etc.) aplicando una operación o filtro a cada elemento. En lugar de usar un bucle tradicional como: <code>cuadrados = [] for x in range(10): cuadrados.append(x**2)</code> Puedes escribir: <code>cuadrados = [x**2 for x in range(10)]</code></p> <p>🎓 ¿Qué hace exactamente una <i>list comprehension</i>? Una <i>list comprehension</i> tiene esta estructura general: <code>[expresión for elemento in iterable if condición]</code> Donde:</p> <ul style="list-style-type: none"> • elemento in iterable es el bucle (como en un for) • expresión es lo que se evalúa y se agrega a la lista • if condición (opcional) filtra los elementos <p>◆ ¿Y con diccionarios? Python también permite usar una sintaxis similar para crear diccionarios, llamada <i>dictionary comprehension</i>.</p> <p>Estructura general: <code>{clave: valor for elemento in iterable if condición}</code> Ejemplo 2: Crear un diccionario de cuadrados <code>cuadrados = {x: x**2 for x in range(5)}</code> # Resultado: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}</p> <p>🔍 ¿Por qué usar comprehensions?</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center; width: 15px;"><input checked="" type="checkbox"/></td> <td style="text-align: center;">Más</td> <td style="text-align: right; vertical-align: bottom;">legible</td> </tr> <tr> <td style="text-align: center;"><input checked="" type="checkbox"/></td> <td style="text-align: center;">Más</td> <td style="text-align: right; vertical-align: bottom;">conciso</td> </tr> <tr> <td style="text-align: center;"><input checked="" type="checkbox"/></td> <td style="text-align: center;">Más rápido que un for tradicional en muchos casos (aunque no siempre)</td> <td style="text-align: right; vertical-align: bottom;"></td> </tr> <tr> <td style="text-align: center;"><input checked="" type="checkbox"/></td> <td style="text-align: center;">Idiomático (estilo Python)</td> <td style="text-align: right; vertical-align: bottom;"></td> </tr> </table>	<input checked="" type="checkbox"/>	Más	legible	<input checked="" type="checkbox"/>	Más	conciso	<input checked="" type="checkbox"/>	Más rápido que un for tradicional en muchos casos (aunque no siempre)		<input checked="" type="checkbox"/>	Idiomático (estilo Python)	
<input checked="" type="checkbox"/>	Más	legible											
<input checked="" type="checkbox"/>	Más	conciso											
<input checked="" type="checkbox"/>	Más rápido que un for tradicional en muchos casos (aunque no siempre)												
<input checked="" type="checkbox"/>	Idiomático (estilo Python)												
¿Cómo usó o adaptó la respuesta?	Usé esto para intentar entender el concepto. Luego, le hice más preguntas específicas a mi caso para lograr a el resultado que necesitaba.												
Reflexión crítica	Sumamente útil, la IA aveces explica mejor que algunos videos de YouTube. Con las preguntas followup pude llegar a el resultado en el código (como esto: <code>records_finales[nivel] = [record for record in records_finales[nivel] if record['nombre'] == game_logic.nombre_jugador]</code>)												
Otros	Dije que iba a cerrar la app si daba una mala respuesta como técnica de prompt engineering.												

Archivos de datos

La estructura de los archivos del programa es la siguiente:



La carpeta de src (source) está inspirada en que estos días estuve trabajando con React + Vite, y vi que usaban eso y me pareció intuitivo, por lo que lo usé para el proyecto. Dentro de src está partidas, donde hay 4 archivos json con 3 partidas de cada dificultad. Dentro de esta misma carpeta, está game_logic.py, juego.py y menus.py; cada uno de ellos cumple una función, el primero, para la lógica del juego y verificar que todo funcione (como el backend), el segundo para que el usuario pueda ver el juego y usar los botones (como el frontend), y el tercero, son una compilación de botones usados en juego.py.

También en src hay 3 archivos json que tienen configuración, records y el juego actual de cada jugador.

Fuera de src, están los módulos (solo hay 1). Este módulo era para hacer errores pero fue descontinuado porque descubrí los messageboxes. En la carpeta de documentación van a estar estos documentos y afuera de todo está el .gitignore y README.md. Estos archivos son para Git, es decir: el primero, es por si tuviéramos variables de entorno, como claves API o así, pero no tenemos. El README.md es como la descripción de nuestro programa, y está en formato Markdown.

Dato importante, el __init__.py no tiene nada, pero está hecho para poder traversar mejor los archivos usando el módulo sys (es sumamente útil).

Conclusiones del trabajo

Problemas encontrados y soluciones a los mismos:

Orden al programar:

Al ser un programa que tomaba bastante más tiempo de realizar, y tenía muchas cosas diferentes como stacks, diccionarios, múltiples archivos, etc; saber que hacer se me complicó un poco, es por ello que opté por descargar una aplicación llamada Todoist, para poder organizar que faltaba de hacer en el programa para hacerlo, y tener un control de que iba haciendo.

Documentación (otra vez):

Me ocurrió otra vez que dejé la documentación para el final, por lo que perdí mucho tiempo documentando las funciones que iba haciendo, para la próxima, tengo que hacer la documentación conforme escribo el código.

Aprendizajes obtenidos:

Uso correcto de tkinter:

Considero que hasta ahora sentí un “click” con tkinter, realmente no me considero buenísimo usando el módulo pero siento que puedo hacer muchas cosas que antes no podía hacer, de manera más eficiente; por ejemplo, usar iteración en tkinter, no tener que hacer variables siempre, etc.

Uso adecuado de commits de Git:

Me pasaba mucho antes del proyecto que los commits que hacía en Git tenía errores, y luego creaba otro para compensar a los errores. Claramente, esto es una mala práctica por lo que lo corregí para este proyecto. Todos los commits tienen cambios importantes, y sin errores.

Estadística de tiempo:

Actividad Realizada	Horas
Análisis del problema	0.5
Diseño de algoritmos	0.25
Investigación de distintos temas	5
Programación	25
Documentación interna	0.5
Pruebas	1
Elaboración del manual de usuario	2
Elaboración de documentación del proyecto	2
Etc (tiempo aprendiendo sobre temas aparte del proyecto).	0.5
TOTAL	40.5

Lista de revisión del proyecto (próxima página):

Concepto	Puntos	Avance 100% / 0	Puntos obtenidos	Análisis de resultados
Opción Jugar: despliegue ventana de juego	15	100	15	
Seleccionar partida	5	100	5	
Botón Iniciar Juego	15	100	15	
Creación del archivo de Récords	5	100	5	
Borrar casillas	2	100	2	
Botón Deshacer Jugada	8	100	8	
Botón Rehacer Jugada	8	100	8	
Botón Borrar Juego	2	100	2	
Botón Terminar Juego	2	100	2	
Botón Records	8	100	8	
Botón Guardar Juego	5	100	5	
Botón Cargar Juego (incluye el despliegue del mismo)	10	100	10	
Opción Configurar	5	100	5	
Ayuda en el programa: Manual de usuario	5	100	5	
Cronómetro o Temporizador en tiempo real	5	100	5	
TOTAL	100	100	100	
Partes desarrolladas adicionalmente				