Índice

Índice de figuras

Introducción

He creado un programa de consola en el lenguaje Python que simula de manera simplificada la gestión de la logística de suministros espaciales. El programa se realiza con el fin de evaluar mi habilidad individual para planear y desarrollar un programa.

Con el programa creado se pueden crear cohetes para luego utilizarlos en los lanzamientos planeados. Se pueden crear peticiones que el mismo programa puede asignarlas a lanzamientos mediante una funcionalidad del mismo. También se permite simular el transcurso de los días viendo como los lanzamientos y sus peticiones son entregados. Toda la información manejada se guarda en archivos y está la funcionalidad de poder crear copias de seguridad, cargarlas o borrarlas.

He realizado un estudio sobre las especificaciones de mi proyecto, después he realizado un borrador en pseudocódigo y finalmente he procedido a implantar mi solución en Python. He dividido el programa en 5 módulos, “core.py”, “data.py”, “file_management.py”, “main.py” y “utils.py”. “Core.py” es el corazón del programa, es dónde convergen todas sus funcionalidades, “data.py” es dónde se encuentran todas las variables de ámbito global, como lo son cadenas de texto para los menús, mensajes de error y las colecciones utilizadas para almacenar los datos del programa. “file_management.py” contiene todas las funciones pertinentes para realizar operaciones con los ficheros, “main.py” es el punto de entrada del programa que contiene el bucle principal del mismo y “utils.py” contiene funciones auxiliares cuyo uso es reiterado durante todo el proyecto.

Después de haberlo planeado todo, he obtenido un resultado satisfactorio y he cumplido con todos los requisitos planteados por el profesor para este proyecto llegando incluso a añadir alguna funcionalidad extra, como es el guardado automático o guardar la fecha del programa incluyéndola en la información histórica también.

1. Objetivos

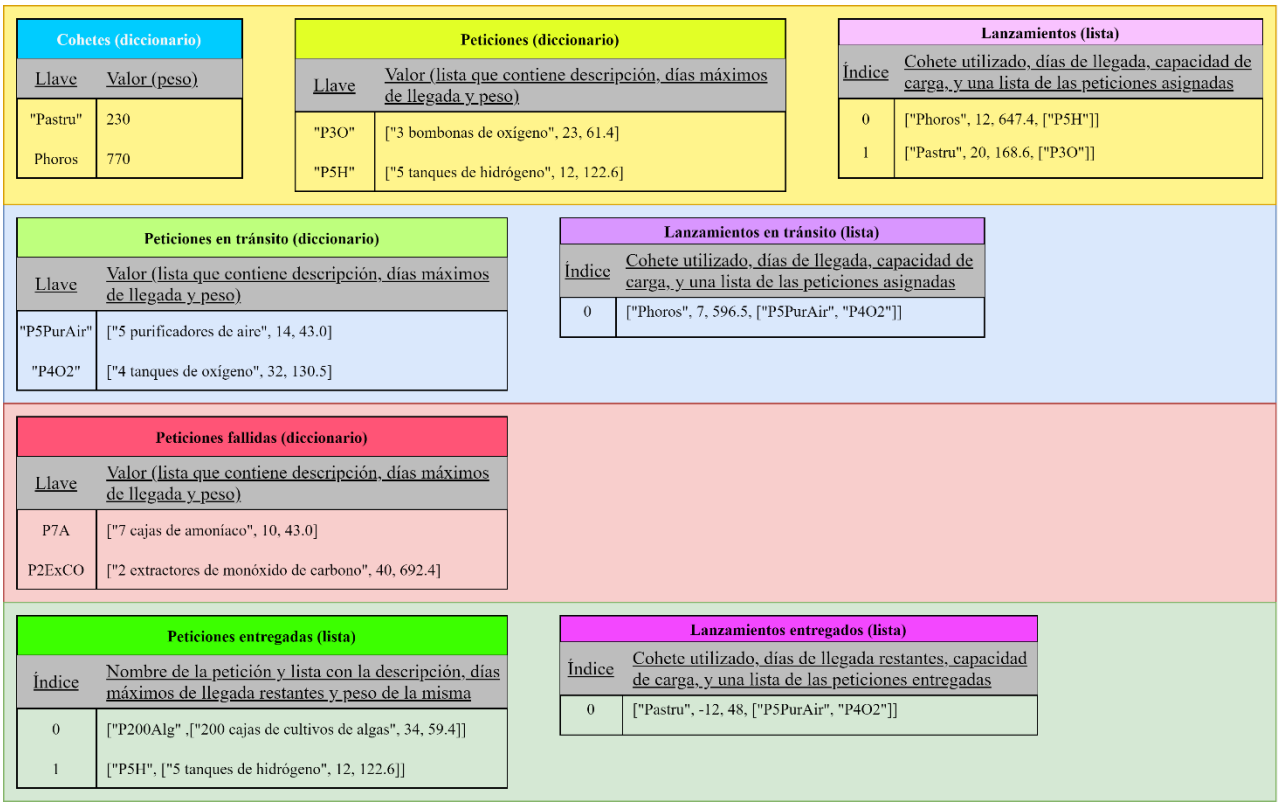
El objetivo del programa es adquirir experiencia para planificar y desarrollar un software desde su inicio hasta su etapa de mantenimiento. En este caso, se trata de un programa de consola en Python que servirá para dotarse a uno mismo de confianza y conocimientos para futuros proyectos.

Los requisitos del programa son los siguientes:

1. Poder crear cohetes con los siguientes datos:
    - Un identificador alfanumérico (nombre), debe comenzar por una letra y tener entre 3-20 caracteres de longitud.
    - Capacidad de carga (en kg), debe ser un número entero mayor que 0.
    - ❖ Ejemplos:
        - ➢ Nombre: “Phoros3”, Capacidad de carga (en kg) 670
- Poder crear peticiones con los siguientes datos:
    - Un identificador alfanumérico (nombre), debe comenzar por una letra y tener entre 3-20 caracteres de longitud.
    - Descripción alfanumérica, debe comenzar por una letra y tener mínimo 8 caracteres de longitud.
    - Peso (en kg), debe ser un número real mayor que 0
    - Días máximos de llegada, deben ser un número entero mayor que 0
    - ❖ Ejemplos:
        - ➢ Nombre: “P3O”, Descripción: “Suministros de 3 tanques de oxígeno”, Peso (en kg): 60.4, Días máximos de llegada: 23
- Poder crear lanzamientos con los siguientes datos:
    - Cohete a utilizar, de los creados previamente
    - Días de llegada, debe ser un número igual o mayor que 0, en caso de 0 el día de llegada sería el mismo día
2. Poder asignar cada petición a un lanzamiento con los siguientes criterios:
    - Primero se deben asignar las peticiones más urgentes, con menos días máximos de llegada
    - Los días de llegada del lanzamiento deben ser inferiores a los días máximos de llegada de la petición
    - Los cohetes de los lanzamientos pierden tanta capacidad de carga como peso de la petición asignada
    - Deben mostrarse por pantalla todos las asignaciones exitosas y fallidas que se realizan
3. Poder simular el transcurso de los días con los siguientes criterios:
    - Se debe introducir mínimo 1 día a transcurrir
    - Todos los lanzamientos que tengan asignada una carga son lanzados y dejan de estar disponibles para la asignación de nuevas peticiones
    - Una vez llegan a la estación se marcan los lanzamientos y sus peticiones como entregados y se guardará un histórico
    - Deben mostrarse por pantalla todos los cambios de estado que se realizan

2. Metodología

Para abordar este problema he diseñado y determinado las colecciones que necesitaba teniendo en cuenta todas las funcionalidades del programa. A continuación, se muestran todas las colecciones como tablas rellenadas de información de ejemplo:



Cohetes (diccionario)

Llave	Valor (peso)
"Pastru"	230
Phoros	770

Peticiones (diccionario)

Llave	Valor (lista que contiene descripción, días máximos de llegada y peso)
"P3O"	["3 bombonas de oxígeno", 23, 61.4]
"P5H"	["5 tanques de hidrógeno", 12, 122.6]

Lanzamientos (lista)

Índice	Cohete utilizado, días de llegada, capacidad de carga, y una lista de las peticiones asignadas
0	["Phoros", 12, 647.4, ["P5H"]]
1	["Pastru", 20, 168.6, ["P3O"]]

Peticiones en tránsito (diccionario)

Llave	Valor (lista que contiene descripción, días máximos de llegada y peso)
"P5PurAir"	["5 purificadores de aire", 14, 43.0]
"P4O2"	["4 tanques de oxígeno", 32, 130.5]

Lanzamientos en tránsito (lista)

Índice	Cohete utilizado, días de llegada, capacidad de carga, y una lista de las peticiones asignadas
0	["Phoros", 7, 596.5, ["P5PurAir", "P4O2"]]

Peticiones fallidas (diccionario)

Llave	Valor (lista que contiene descripción, días máximos de llegada y peso)
P7A	["7 cajas de amoníaco", 10, 43.0]
P2ExCO	["2 extractores de monóxido de carbono", 40, 692.4]

Peticiones entregadas (lista)

Índice	Nombre de la petición y lista con la descripción, días máximos de llegada restantes y peso de la misma
0	["P200Alg" ,["200 cajas de cultivos de algas", 34, 59.4]]
1	["P5H", ["5 tanques de hidrógeno", 12, 122.6]]

Lanzamientos entregados (lista)

Índice	Cohete utilizado, días de llegada restantes, capacidad de carga, y una lista de las peticiones entregadas
0	["Pastru", -12, 48, ["P5PurAir", "P4O2"]]

Figura 1. Colecciones del programa

Para guardar los cohetes creados he elegido utilizar un diccionario ya que es más cómodo para acceder a sus valores y porque me da la seguridad de que nunca van a poder haber dos cohetes con el mismo nombre. También quiero que las peticiones sean únicas, por tanto, elegí un diccionario también. Para los lanzamientos he elegido una lista ya que no hace falta un identificador único para ellos y porque puede haber varios lanzamientos con las mismas especificaciones.

En el diccionario de los cohetes se va a guardar el nombre del cohete como la llave y el peso como el valor. En el caso de las peticiones se guardará el nombre de la petición como llave y el valor será una lista que contiene la descripción, días máximos de llegada y peso de la misma. En el caso de los lanzamientos se añadirá a la lista de todos ellos una lista conteniendo el cohete utilizado, los días de llegada, la capacidad de carga y una lista con las peticiones asignadas a dicho lanzamiento.

Además de esto, hay otras colecciones auxiliares: “peticiones en tránsito”, “peticiones fallidas”, “lanzamientos en tránsito”, “peticiones entregadas” y “lanzamientos entregados”. Éstas servirán para determinar el estado en el que se encuentran las peticiones y lanzamientos.

Después he hecho un diagrama general de cómo debía ser el programa. En el punto de entrada del programa habría un bucle para el menú principal con opciones a elegir para operar el programa, y dentro de cada opción habría otro bucle con el submenú correspondiente conteniendo todas las acciones que se puedan tomar con dicha operación.

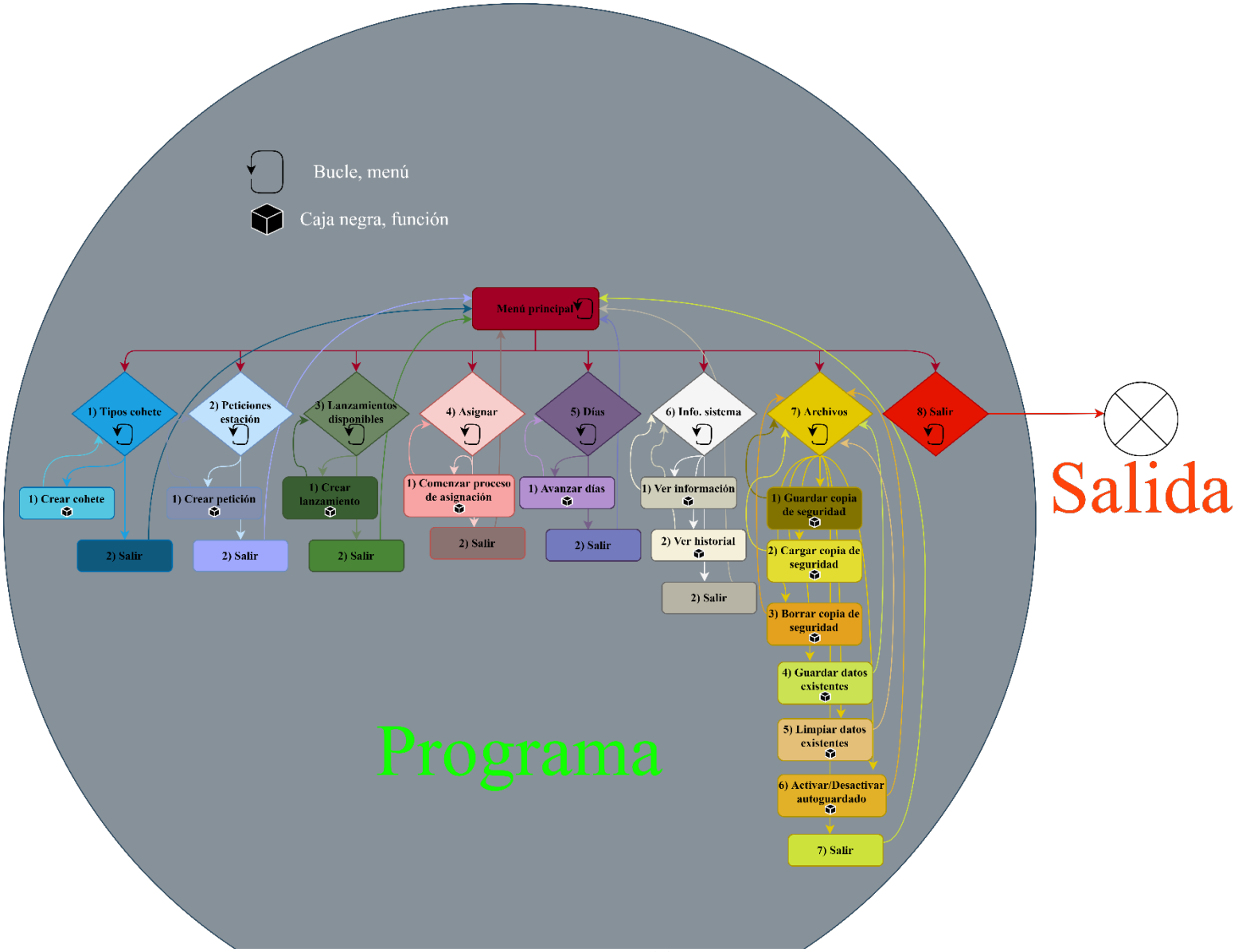


Figura 2. Diagrama del programa

Con todo esto en mente, he comenzado por definirme los módulos que necesitaría. Después he comenzado por el punto de entrada, he creado el bucle del menú general y a partir de ahí me he puesto manos a la obra.

Después de haber creado el módulo “main.py”, he comenzado con el módulo “data.py”; creé las colecciones de datos que precisaba el programa y comencé a hacer las cadenas de texto correspondientes a los menús, a los mensajes de error, etc.

El paso natural siguiente fue empezar a desarrollar el corazón del programa, el módulo “core.py”. Comencé pensando en cómo hacer un sistema de elección robusto y a prueba de errores para cada menú. Conforme iba escribiendo empecé a darme cuenta de que me faltaba otro módulo para las funciones auxiliares y usadas de forma reiterada.

Aquí fue dónde nació el módulo “utils.py”, en él empecé a escribir las funciones para recoger las respuestas del usuario, para imprimir mensajes con varias opciones en pantalla, para mostrar los menús, para validar respuestas y algunas más.

El proceso consistía en ir haciendo cada funcionalidad del menú en orden ascendente, comenzando por “1) Tipos cohete” hasta “7) Archivos”. Cuando terminaba de escribir una funcionalidad hacía unas cuántas pruebas, veía que iba mal y que iba bien, y decidía si valía la pena estancarse para arreglarlo o seguir adelante y volver más tarde. En función de una cosa u otra, siempre me dejaba notas “#TODO” para estar al tanto de todo.

Tras varias iteraciones finalmente me quedaba solamente la parte tediosa de los ficheros. Decidí emplear json por su facilidad de uso y el buen equipo que hace con Python. Aquí tuve que crear otro módulo “file_management.py” y determinar el rango de esta última funcionalidad. Cuando lo tuve claro comencé a escribir el código esencial para la creación de carpetas y ficheros, y después tuve que trabajar en paralelo en el módulo “core.py” y “file_management.py” coordinando el código entre ambos para poder lograr mi cometido.

Después de unas cuantas pruebas intensivas me di cuenta de varios bugs y varias especificaciones que no cumplía el programa y tuve que hacer unas cuantas iteraciones más sobre todos los módulos con el fin de solucionar las cosas. En este proceso también hice uso intensivo de datos “hard-coded” para agilizar las tareas de testeo.

Durante el proceso eché de menos una herramienta para hacer pruebas unitarias y en algunas ocasiones no haber trabajado con herramientas de control de versiones.

Dicho esto, el proyecto tampoco tenía una dimensión tan grande para poder sacarle todo el jugo a las herramientas comentadas.

3. Resultados

Después de haber aplicado toda la metodología explicada he comenzado haciendo un primer borrador del programa con funcionalidades muy básicas. El proceso ha sido iterativo a partir de ahí implementando las funcionalidades una a una de manera pertinente. Me he centrado primero en las partes funcionales del programa que dependen de los datos del mismo, y he dejado la parte de almacenar los datos de manera externa para la parte final.

El menú implementado en Python ha quedado de la siguiente manera:

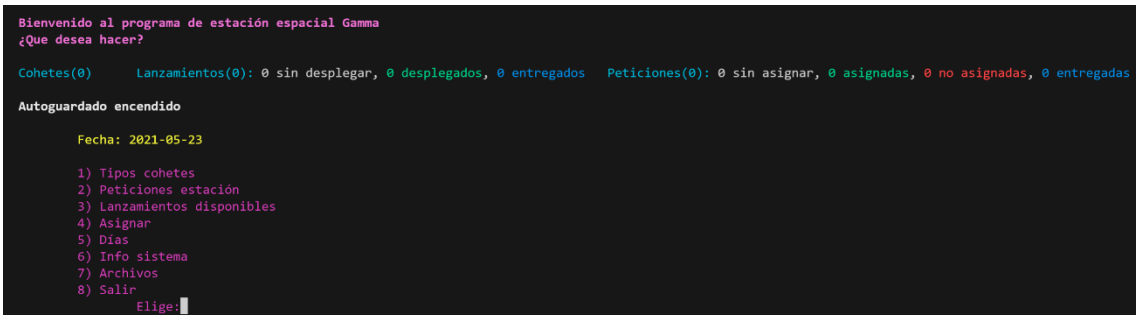


Figura 3. Menú del programa

Comentaré brevemente en que consiste el flujo de ejecución del programa. Como se ha comentado el programa tiene como punto de entrada el archivo "main.py". Este archivo importa varios módulos para poder operar:

- "core.py", el corazón de todo el programa. En él se encuentran todas las funcionalidades implementadas y toda la lógica de todas ellas.
- "utils.py", el módulo de las funciones auxiliares. En este archivo se encuentran las funciones con uso general y reiterado como son imprimir en pantalla con diferentes estilos o mensaje de espera, recoger respuesta del usuario, verificar si la respuesta es válida, etc.
- "data.py", el fichero que contiene todos los datos estáticos y dinámicos del programa. Contiene todos los mensajes que se imprimen en pantalla y las colecciones que se utilizan durante todo el programa.

En "main.py" es dónde se encuentra el gran bucle del menú principal.

Al importar el archivo "core.py" desde "main.py", este primero también importa a su vez "file_management.py". Este último módulo hace las comprobaciones pertinentes de las carpetas que necesita para poder operar, y en caso de no existir se encargara de crearlas. En caso de no tener definido el autoguardado y la fecha con la que se está ejecutando el programa, el mismo módulo lo definiría.

Ahora procedo a explicar cada operación y funcionalidad del programa en orden ascendente.

1. Tipos cohetes:

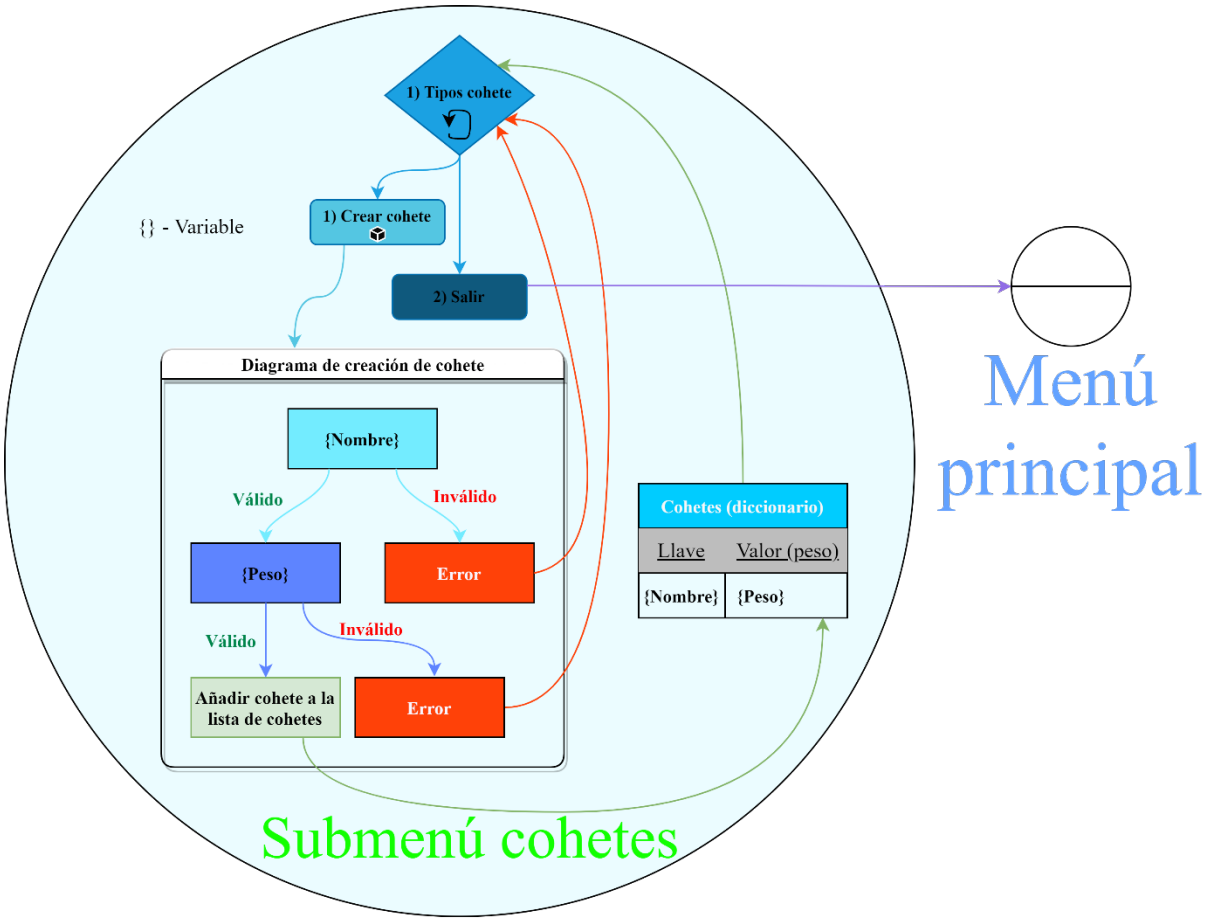


Figura 4. Diagrama de flujo submenú cohetes

En este submenú existen 2 acciones a tomar:

- Crear cohete: aquí se le pide al usuario que introduzca los datos del cohete en el orden en el que está indicado en el diagrama. Si alguno de los datos que se piden no cumple los requisitos correspondientes se le notificará al usuario y se volverá al submenú otra vez. Si todo es introducido adecuadamente se añadirá el nuevo cohete al diccionario de los cohetes.
- Salir: esta acción regresará al usuario de vuelta al menú principal.

Los submenús de las peticiones y los lanzamientos son exactamente iguales, por tanto, no voy a insistir con unos diagramas casi calcados para ellos dos. También el diagrama de flujo para crear peticiones y lanzamientos es el mismo, solamente habría que añadir algunos campos más recoger la información específica de cada uno de ellos.

4. Asignar:
   En este submenú existen 2 acciones a tomar:
   - Comenzar proceso de asignación: primero de todo se ordenan las peticiones y los lanzamientos de manera ascendente en función del número de días. Después hay que emparejar las peticiones con los lanzamientos, para eso se va a emplear un bucle anidado dentro de otro. El bucle exterior recorrerá las peticiones y el interior los lanzamientos. Entonces, por cada petición se comprobará que lanzamiento es el que cumple los requisitos para asignársela, es decir, que lanzamiento tiene el mismo o menor número de días que la petición y mayor o igual peso disponible.
     Si no se encuentra ninguno, la petición será eliminada del diccionario de las peticiones y será guardada en el diccionario de las peticiones fallidas.
     Si se encuentra un lanzamiento compatible, la petición será eliminada del diccionario de las peticiones y será guardada en el diccionario de las peticiones en tránsito. Asimismo, se guardará el nombre de la petición en la lista de las peticiones asignadas del lanzamiento correspondiente. Cada vez que sea posible asignarle una petición a un lanzamiento se tendrá que descontar el peso de dicha petición a la capacidad de carga del lanzamiento.
     Después de haber emparejado las peticiones con los lanzamientos, hay que emparejar las peticiones fallidas, si las hubiera, de la misma manera que hemos hecho hasta ahora. En caso de seguir sin encontrar un lanzamiento compatible no habría que hacer nada esta vez, y en caso de éxito se eliminarían del diccionario de las peticiones fallidas y se añadirían al diccionario de las peticiones en tránsito.
   - Salir: esta acción regresará al usuario de vuelta al menú principal.
     Un ejemplo de pseudocódigo para esta funcionalidad sería el siguiente:

```
peticiones = ordenar_por_dias(peticiones, ascendente=si)
lanzamientos = ordenar_por_lanzamientos(lanzamientos, ascendente=si)

for i = 0:longitud(peticiones) {
    asignado = False
    for j = 0:longitud(lanzamientos) {
        si peso(peticiones[i]) == peso(lanzamientos[j]) {
            si dias(peticiones[i]) == dias(lanzamientos[j]) {
                restar_peso(peticion[i], lanzamientos[j])
                asignar_peticion_a_lanzamiento(peticion[i], lanzamientos[j])
                añadir_peticion_en_transito(peticiones[i])
                eliminar_peticion(peticiones[i])
                asignado = True
            }
        }
    }
    si !asignado {
        añadir_peticion_fallida(peticiones[i])
        eliminar_peticion(peticiones[i])
    }
}
```

Figura 5. Pseudocódigo funcionalidad asignar

Veamos unos ejemplos de esta funcionalidad:

- Información antes de asignar

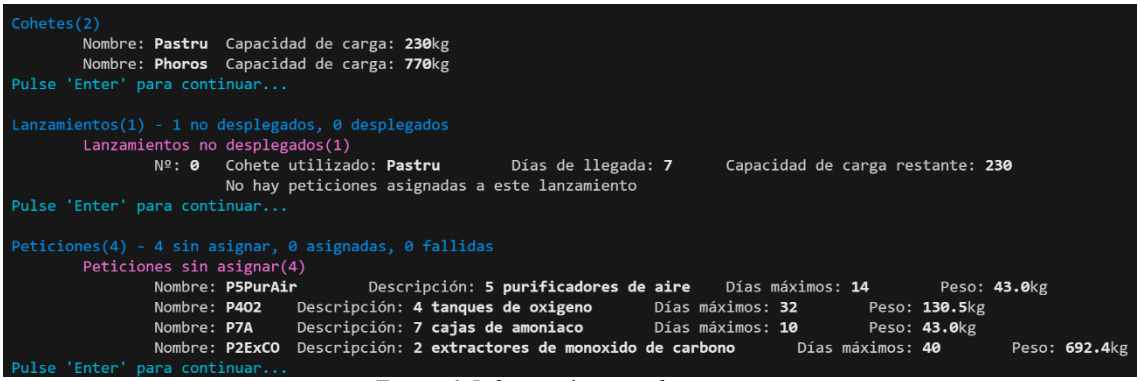


Figura 6. Información antes de asignar

- Proceso de asignación

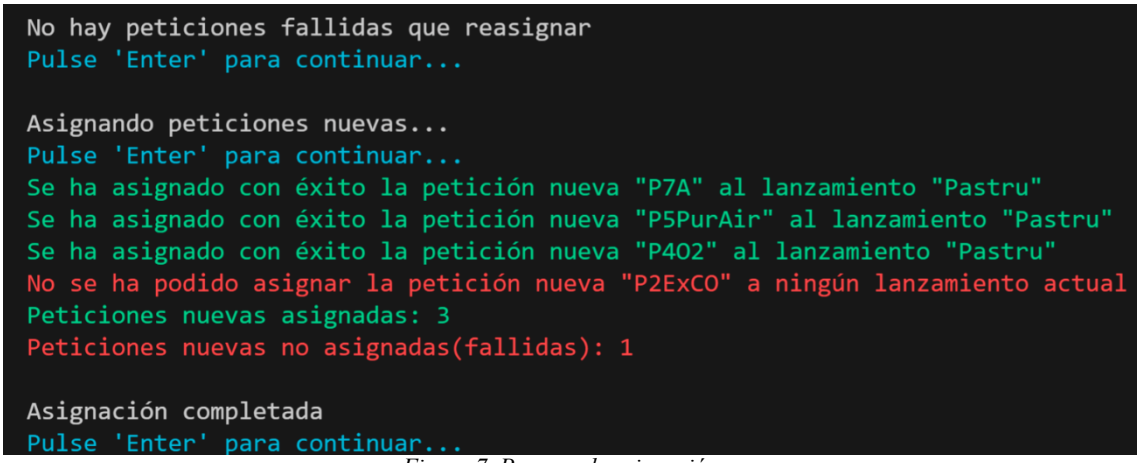


Figura 7. Proceso de asignación

- Información después de asignar

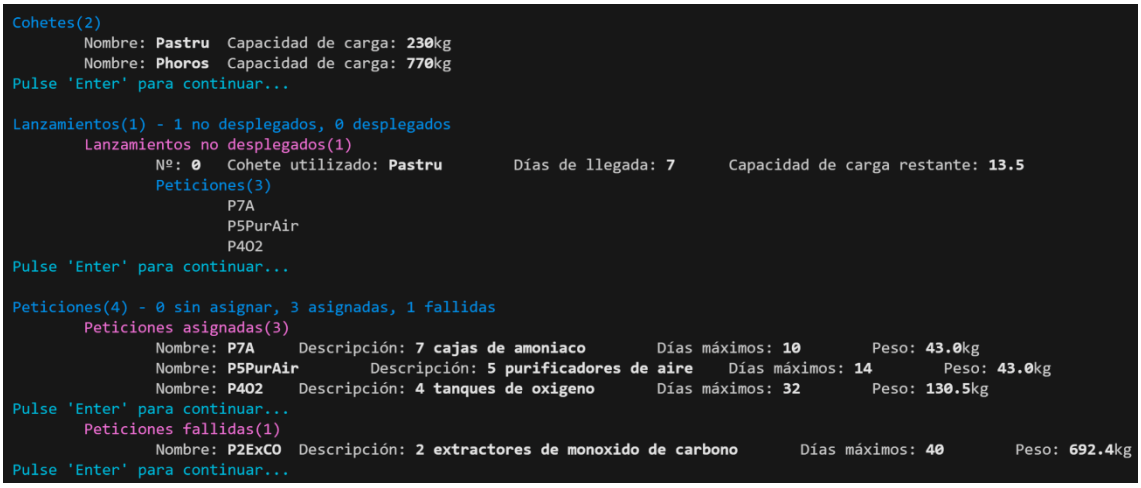


Figura 8. Información después de asignar

5. Días:
   En este submenú existen 2 acciones a tomar:
   - Avanzar días: esta acción simulará el paso de los días. Se le pedirá al usuario el número de días a transcurrir; seguidamente se podrán en tránsito todos los lanzamientos con peticiones, es decir, se quitarán de la lista de lanzamientos y se añadirán a la lista de lanzamientos en tránsito.
     Después se iterará sobre la lista de los lanzamientos en tránsito y se restarán los días a transcurrir a cada uno. A continuación, se comprobará si los días de llegada del lanzamiento iterado es igual o menor que 0.
     El caso afirmativo significa que el lanzamiento ha sido entregado; por tanto, habría que quitarlo de la lista de lanzamientos en tránsito y añadirla a la lista de lanzamientos entregados. Se iterará seguidamente sobre cada una de sus peticiones eliminándolas del diccionario de las peticiones en tránsito y añadiéndolas a la lista de las peticiones entregadas. Como funcionalidad extra, se sobrescribirá el campo de los días de llegada del lanzamiento entregada con la fecha de entrega. Todo esto irá acompañado de un correcto feedback para que el usuario sepa en todo momento lo que está pasando.
     Un posible pseudocódigo para esta funcionalidad sería el siguiente:

```
dias = input("Introduzca los dias a transcurrir")
respuesta_valida = comprobacion_de_input(dias)
si !respuesta_valida{
    error("Debes introducir un número mayor que 0")
    return
}

dias = string_a_entero(dias)

for i = 0:longitud(lanzamientos) {
    si tiene_peticiones(lanzamientos[i]) {
        añadir_lanzamiento_en_transito(lanzamientos[i])
    }
}

for i = 0:longitud(lanzamientos_en_transito) {
    dias(lanzamientos_en_transito[i]) = dias(lanzamientos_en_transito[i]) - dias
    if (lanzamientos_en_transito[i]) < 1 {
        for j in 0:longitud(peticiones(lanzamientos_en_transito[i])) {
            añadir_peticion_entregada(peticiones[lanzamientos_en_transito][j])
            eliminar_peticion_en_transito(peticiones[lanzamientos_en_transito][j
        }
        añadir_lanzamiento_entregado(lanzamientos_en_transito[i])
        eliminar_lanzamiento_en_transito(lanzamientos_en_transito[i])
    }
}
```

Figura 9. Pseudocódigo funcionalidad días

   - Salir: esta acción regresará al usuario de vuelta al menú principal.

Veamos unos ejemplos de esta funcionalidad:

- Información antes de transcurrir días

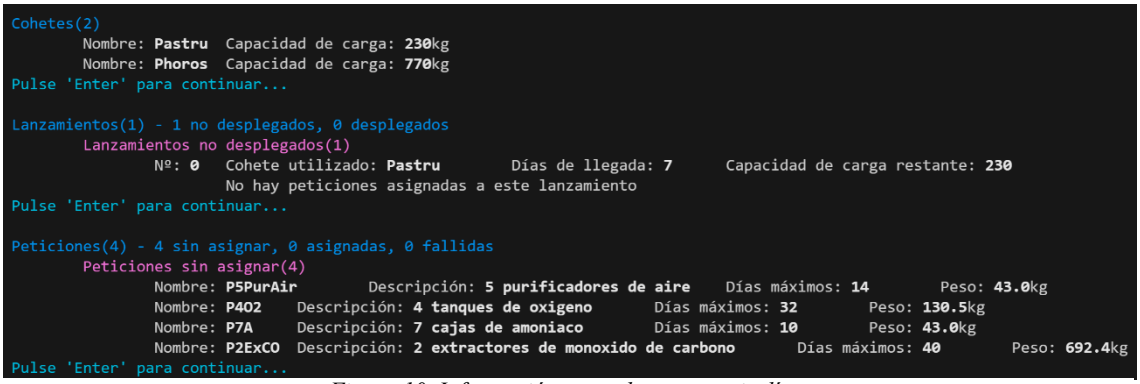


Figura 10. Información antes de transcurrir días

- Proceso de transcurrir días

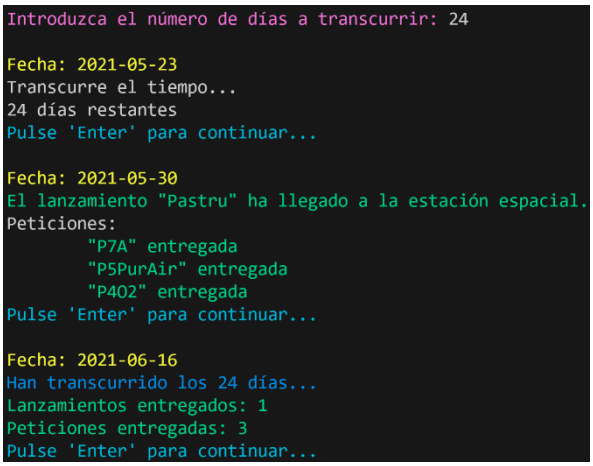


Figura 11. Proceso de transcurrir días

- Información después de transcurrir días

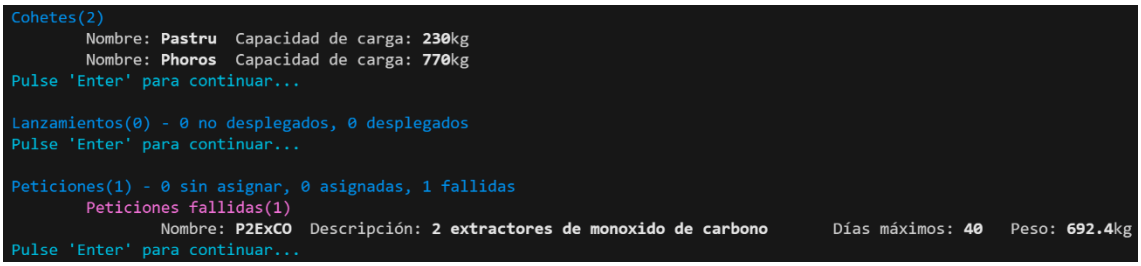


Figura 12. Información después de transcurrir días

- Historial después de transcurrir días

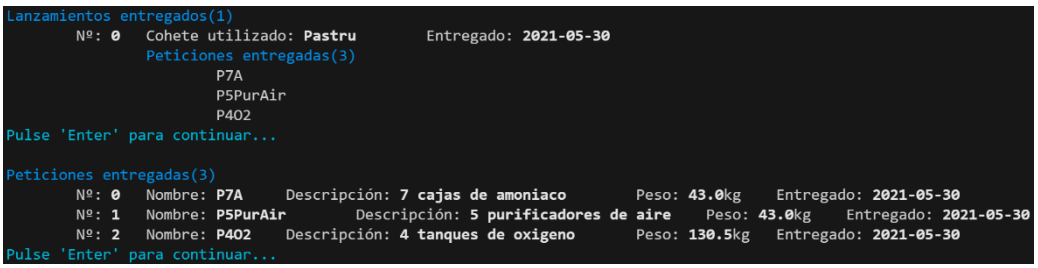


Figura 13. Historial después de transcurrir días

6. Info sistema:
   En este submenú existen 3 acciones a tomar:
   - Ver información: consiste en la impresión a consola de toda la información respecto a los cohetes, peticiones y lanzamientos que no están entregados aún.
     El código utilizado consiste solamente en bucles que van iterando sobre las colecciones de los cohetes, peticiones, peticiones en tránsito, peticiones fallidas, lanzamientos y lanzamientos en tránsito. En esos bucles se formatea la impresión en consola de la información para que quede clara y haciendo las pausas necesarias para que el usuario no se pierda.
     Ejemplo:

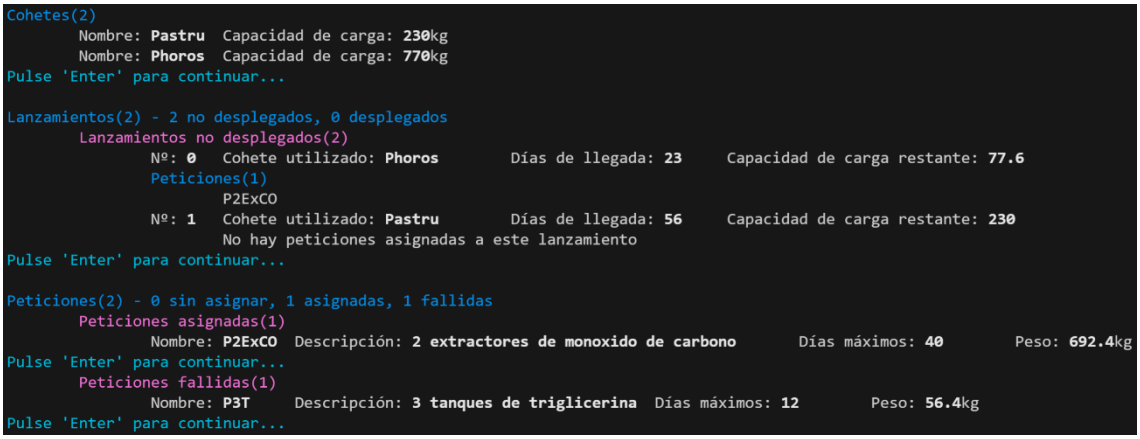


Figura 14. Información general

   - Ver historial: hace la misma función que la mencionada anteriormente pero con las peticiones y lanzamientos entregados.
     Ejemplo:

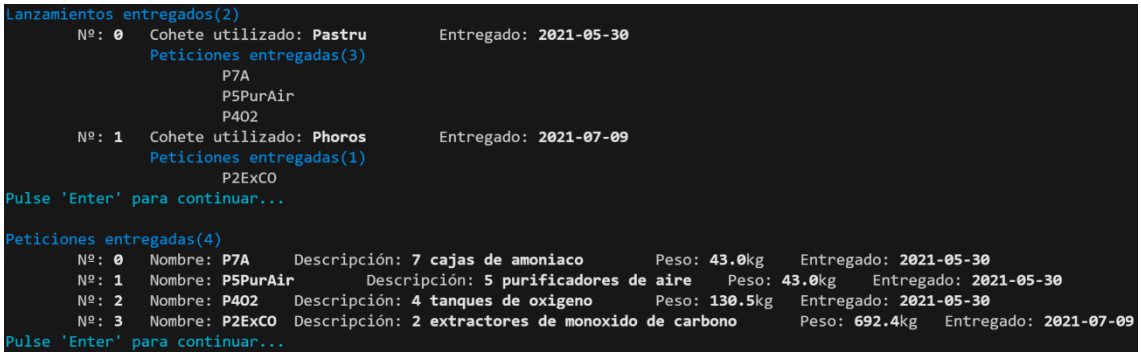


Figura 15. Historial

   - Salir: esta acción regresará al usuario de vuelta al menú principal.

7. Archivos:
   En este submenú existen 7 acciones a tomar:
   - Guardar copia de seguridad: genera un fichero nombrado con el formato de la fecha actual (año/mes/día/hora/minuto/segundo) con la extensión “json”. Se guarda toda la información actual haciendo uso de funciones del módulo “file_management.py”. Primero se compone toda la información en un diccionario global y después este diccionario es pasado a otra función junto con el nombre del archivo de la copia para ser creados y guardados.
   - Cargar copia de seguridad: muestra al usuario una lista de todas las copias de seguridad existentes y se le pide elegir la que quiere cargar en los datos del programa. Esta funcionalidad también hace uso intensivo del módulo “file_management.py”. Mediante una función del mismo se abre el archivo elegido, se vuelca todo el diccionario global en una variable equivalente a un diccionario de Python. Después mediante un bucle se recorre el diccionario y se va asignando los diferentes valores de las llaves a sus correspondientes colecciones, los cohetes del fichero con los cohetes del programa, las peticiones del fichero con las peticiones del programa, etc.
   - Borrar copia de seguridad: muestra al usuario una lista de todas las copias de seguridad existentes y se le pide elegir la que desea borrar. Obvia decir otra vez que aquí se utiliza de nuevo el módulo “file_managament.py”; mediante una función suya se borra el archivo que el usuario ha indicado.
   - Guardar datos existentes: guarda todos los datos del programa en dos archivos “general.json” y “historic.json”. Estos archivos son los que se cargan automáticamente al abrir el programa. El archivo “historic.json” guarda la información de las peticiones y los lanzamientos entregados, y “general.json” guarda la información de las demás colecciones utilizadas en el programa.
   - Limpiar datos existentes: vacía todas las colecciones del programa. Esta funcionalidad y la de antes hacen uso del módulo “file_management.py” para alcanzar estas tareas.
   - Activar/Desactivar autoguardado: activa o desactiva esta funcionalidad. Se trata de poner solamente la variable correspondiente al autoguardado en “True” o en “False”. Esta variable se guarda solamente en el archivo de carga automática “general.json”.
   - Salir: esta acción regresará al usuario de vuelta al menú principal.

4. Conclusiones

Tras haber finalizado el desarrollo y mantenimiento del programa he de decir que sienta bien simplemente. Ha habido muchos momentos de frustración y estancamiento, pero al final he dado con la solución siempre.

Creo que me ha servido mucho para aprender a planear proyectos y evitar los errores que he cometido realizando este. Hay que saber elegir las herramientas adecuadas para cada uno encontrando el equilibrio entre las utilidades de la herramienta y el tiempo de dedicación a la misma, es decir, valorar la rentabilidad de cada herramienta que se vaya a utilizar para el proyecto. Debo decir que, en mi caso, ha sido un error no haber utilizado una herramienta de control de versiones porque he hecho algunas modificaciones que resultaron ser erróneas más adelante y quisiera haber podido tener la antigua versión de dicha modificación.

Durante el desarrollo de esta actividad me he familiarizado mucho con metodologías de trabajo como el brainstorming, el hacer diagramas de flujo, elaborar pseudocódigo, hacer sketches, utilizar breakpoints para depurar el programa, etc.

Debo decir que la experiencia ha sido muy enriquecedora para mis conocimientos y me han ayudado a tener más perspectiva para futuros proyectos.

5. Referencias

areatecnologia. (s.f.). Obtenido de https://www.areatecnologia.com/diagramas-de-flujo.htm

Marker, G. (s.f.). *tecnologia-informatica.* Obtenido de https://www.tecnologia-informatica.com/pseudocodigo/

Python. (s.f.). Obtenido de https://www.python.org/doc/

StackOverFlow. (s.f.). Obtenido de https://stackoverflow.com/questions/tagged/python

w3schools. (s.f.). Obtenido de www.w3schools.com