Py.Karel

1) Equipo: Juan Andrés Alayón Ariza, Wenceslao Huang, Jaider Duván Velasco Díaz.
Repositorio GitHub: https://github.com/GH-Jaider/Py.Karel#pykarel

2) Resumen ejecutivo:

El problema identificado fue que al trabajar con Karel en el primer corte del curso de programación de computadores notamos que el lenguaje no tenía un interpretador que nos fuese útil a nosotros para poder identificar errores y como se da la ejecución de nuestro código. Por lo tanto, necesitábamos tener un lápiz y papel a mano para poder corroborar paso por paso nuestro código y buscar errores en caso que lo haya, siendo susceptibles a equivocarnos verificando la ejecución. Para los profesores también era un gran trabajo corregir quices o parciales, ya que cada alumno programa de distinta manera y para encontrar la solución se podían escribir distintos algoritmos. Con este problema identificado nosotros decimos crear un intérprete para el lenguaje Karel que aprendimos, notificando errores cuando los haya y ejecutandolo visualmente para que el usuario verifique su algoritmo.

3) Funcionalidad de la solución computacional:

Py.karel es una interfaz que recibe dos archivos en formato ".txt" y permite visualizar la ejecución de instrucciones para Karel. Uno de los archivos es el código, que debe estar escrito de manera correcta y con las tabulaciones necesarias, además de seguir las instrucciones declaradas en la interfaz. Y el otro es el mapa de Karel en formato ASCII. Una vez que estos dos archivos se encuentran en la carpeta de ejecutables se inicia el programa que lee el archivo con las instrucciones y el archivo de mapa, luego se muestra la ejecución en la interfaz visual, si la ejecución se da de manera correcta un mensaje aparece informando, de lo contrario un mensaje le dice al usuario que tiene errores y le sugiere revisar el documento de las instrucciones.
Con esta solución se le da la posibilidad a nuevos estudiantes de MACC de verificar sus algoritmos para Karel y ver cómo se ejecutan, además de permitirle a los profesores evaluar de una manera más rápida y sencilla.

4) Objetivos específicos:

1. Utilizar los conocimientos adquiridos en el curso de programación de computadores para desarrollar un programa: Se logró aplicar lo aprendido e investigado para desarrollar e implementar una interfaz que interpreta instrucciones para Karel y muestra al usuario la ejecución.

2. Implementar un diseño atractivo para la interfaz de la ejecución de Karel: Consideramos que se logró. Desarrollamos un pequeño branding en torno al programa, seleccionando una paleta de colores y creando un logo e ilustraciones para mostrar y hacer más amigable la interfaz.

3. Realizar depuraciones constantes en busca de algún tipo de error: Este apartado fue muy importante para nosotros y nos esmeramos por cubrir cada posible error del programa. Las depuraciones se hacían cada vez que se agregaba o cambiaba algo en el código para verificar la funcionalidad. Objetivo cumplido.

4. Seguir las reglas y utilizar las bases del lenguaje de Karel:
   Desarrollamos py.karel basándonos en el lenguaje de programación lúdico Karel, no obstante en nuestra versión decidimos hacer algunos cambios. Dichos cambios están

incluidos en las instrucciones que son visibles en la interfaz. Entre ellas decidimos no implementar el condicional “ELSE” ni permitir el uso de “IF” o “WHILE” al definir una nueva instrucción o dentro de un “ITERATE”, aun así todos los mapas que trabajamos al aprender Karel son solucionables con py.karel. Dicho esto, se podría decir que este objetivo se cumplió de manera parcial pero completamente funcional.

5. Desarrollar una interfaz que recibe un archivo “.txt” que contiene el código de Karel y mediante ese archivo se ejecuta:
   Nuestra interfaz recibe un archivo “Test.txt” que contiene las instrucciones de Karel en la carpeta Ejecutables, además recibe en la misma carpeta un archivo llamado “mapa.txt” que contiene el mapa de Karel interpretado con caracteres ASCII, en las instrucciones de la interfaz se incluye como transcribir el mapa a formato ASCII. Es así que damos este objetivo por logrado.

5) Herramientas de programación:

En nuestro Proyecto utilizamos el módulo Pygame para la interfaz visual, en ella usamos la implementación de botones, imágenes, texto y el seguimiento de la posición del mouse y la revisión de si se efectúa click. También, para la lectura de código y las definiciones de la posición de Karel, la posición de los beepers y la posición de los muros usamos la apertura y lectura de archivos, la función try / except, listas, tuplas, ciclos, condicionales, módulos, diccionarios y el formato de texto ASCII. De forma adicional fueron usados los módulos “sys” y “os”.
Para desarrollar el código, usamos distintos IDE como Spyder y pycharm, esto por la comodidad de los integrantes del grupo y verificar el funcionamiento desde distintas interfaces.

6) Descripción del programa:

I. Nuestro programa está compuesto por 5 archivos con extensión “.py” y 7 imágenes en formato “.png”. Los archivos son: “colorsandtext”, “ejecicionview”, “funciones”, “lecturamapas” y “Main_Execution”

II. Main_Execution.py es el archivo que contiene el bloque de ejecución de py.karel

III. Las funciones de cada archivo son:

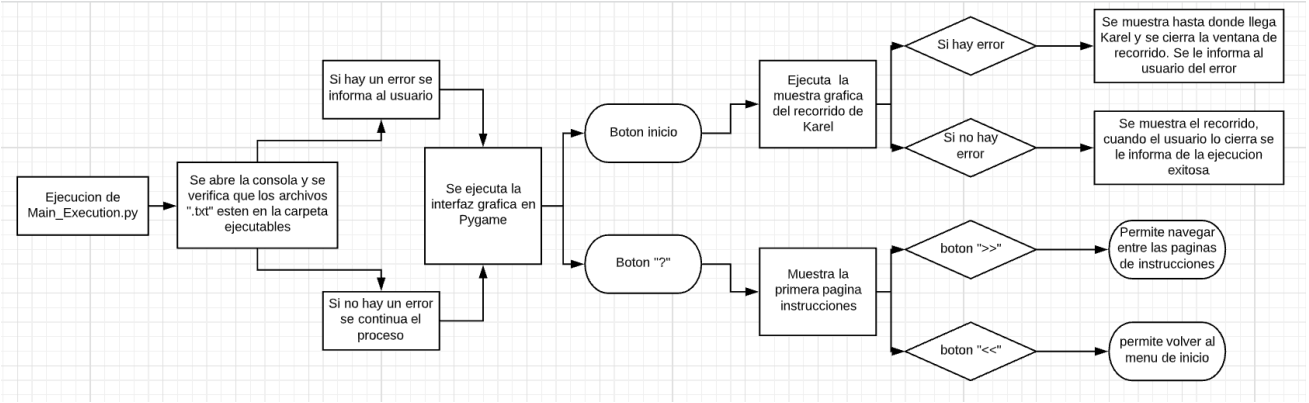
- colorsandtext.py: Contiene tupas con valores RGB que hacen más fácil la implementación en los bloques de ejecución y variables de tipo string con los textos que se implementaron en la interfaz gráfica
- ejecucionview.py: contiene el bloque que ejecuta interpreta la parte logica de lecturamapa.py y la entrega un resultado gráfico que podemos mostrar con pygame
- funciones.py: contiene las funciones que se definieron para la implementación de la parte gráfica del programa; entre ella están: “boton”, “titulos”, “texto”, “textosmall”, “creditos”, “texter”, “s_vertical”, “d_vertical”

- lecturamapas.py: contiene todas las funciones y ejecucion logica de la interpretación de los mapas en formato ASCII y las instrucciones de Karel(se explicará a profundidad más adelante)
    - front_is_blocked, right_is_blocked, left_is_blocked, next_to_a_beeper, move, turnleft, pickbeeper, putbeeper, escrituraCodigo, Ftab, lecturaCodigonotabs, new_instruction_check, new_instruction, boe,

lectura_instrucciones, iterate_check, iterate, End, acción, cond_if_check, condIf, cond_while_check, condWhile, NewBeepr.

- Main_Execution.py: Es el archivo que contiene el bloque de ejecución pygame y lógico que se muestra al usuario, formado por las funciones descritas y funciones propias de pygame.

IV. Orden cronológico de la ejecución:
https://lucid.app/lucidchart/invitations/accept/inv_69f021f8-97df-4e5a-b9e8-fed81d555891?viewport_loc=-25%2C-116%2C1993%2C823%2C0_0



V. Descripción detallada funciones principales:

1. ejecucionview.py: Primero, define el tamaño de la ventana y dibuja los puntos de la cuadrícula. Segundo, dibuja en la cuadrícula la posición inicial de karel, beepers y muros. Tercero, hace una prueba de ejecución del código del usuario donde itera sobre una lista lecturamapas.recorridoSN donde se encuentran todas las instrucciones, a partir de cada valor tomado dibuja lo que se necesita para visualizar la ejecución. Por último, si la prueba es correcta espera a que el usuario cierre la ventana, caso contrario se cerrará automáticamente luego de visualizar la ejecución y aparecera en consola “Hubo un error en la ejecución”.
    - s_vertical: Con la x que recibe se le asigna cada y, se crea una tupla por cada una y la agrega la lista vacía que se creó y luego la retorna
    - north, east, south, west : Cada función define los tres puntos de karel (triángulo) y retorna la variable que contiene en una tupla con 3 tuplas ((x1, y1),(x2, y2),(x3, y3))
    - d_vertical: Con la x que recibe se le asigna cada y, se crea una tupla por cada una y la agrega la lista vacía que se creó y luego la retorna
2. lecturamapa.py:
        - front_is_blocked: revisa la dirección hacia donde está viendo karel mediante la lista facing, si karel está viendo hacia el norte revisa la coordenada de karel con una variante según hacia donde esta viendo, si esa coordenada está en la lista muros, o está fuera del mapa retorna verdadero, si no retorna falso.
        - right_is_blocked: misma lógica que el anterior, pero en cambio la variación es según donde es la derecha de karel.
        - left_is_blocked: igual que los dos anteriores, pero con la izquierda.
        - next_to_a_beeper: revisa si la coordenada de karel está en la lista de de las coordenadas de los beepers, si si retorna verdadero, si no retorna falso.

- move: primero revisa si el frente está bloqueado, esto lo hace llamando a la función front_is_blocked, si esta bloqueado imprime que hay un error e impide al código avanzar más de la cuenta, si no está bloqueado revisa hacia donde está mirando karel y si karel está sobre un beeper, luego de eso reemplaza en la lista del mapa el símbolo de karel por un punto o por un beeper, según si está sobre o no un beeper, después de eso, modifica la coordenada de karel y su punto respectivo en el mapa según donde debería quedar karel después del move y agrega a la lista s_coordenadaKarel la posición en el movimiento, según karel donde está y dónde está viendo .
- turnleft: reemplaza el carácter de karel por el cual representa a karel mirando a la izquierda segun donde está viendo, y posteriormente reemplaza el valor sub 0 de la lista facing según donde este viendo karel luego del turnleft y agrega a la lista s_coordenadaKarel la posición en el movimiento, según karel donde está y dónde está viendo luego del turnleft, además llama a la función NewBeepr
- pickbeeper: revisa si la coordenada de karel está en la lista de las coordenadas de los beepers, si lo esta, quita de esa lista, el beeper donde esta karel, aumenta el número de beepers que tiene karel agrega la posición actual de karel y donde esta viendo a s_coordenadakarel y hace NewBeepr.
- putbeeper: lo mismo que el anterior, solo que en vez de quitarlo de la lista lo agrega, y en vez de aumentar el valor del número de beepers de karel, lo disminuye.
- escrituraCodigo: se encarga en abrir el archivo Test.txt, leerlo, quitarle los \n y transcribir la lista que retorna el leerlo y lo convierte en un diccionaro cuyas llaves son números de 1 a la longitud del código.
- Ftab: revisa si el valor de las tabulaciones del código son las mismas que las pensadas
- lecturaCodigo: se encarga de revisar si el código es BOP, BOE define new instruccion, las acciones, EOE o EOP. Si lo hace, llama a las funciones respectivas para revisar y ejecutar el codigo
- notabs: revisa si es un BOP o un EOP pues son los que no tienen tabulaciones, aumenta las tabulaciones en 1 a aumenta el numero de linea si es un BOP.
- new_instruction_check: Revisa si la línea actual es la definición de una nueva instrucción
- new_instruction: agrega al diccionario de instrucciones el nombre de la nueva instrucción con una lista de las instrucciones que se encuentran dentro del define.
- boe: revisa cuando hay un boe o un eoe y vuelva la condición BOE true
- lectura_instrucciones: Retorna un numero o una lista segun la instrucción que se le de. (es una lista si es una nueva instruccion definida)
- iterate_check: Revisa si es un itrate y retorna el número de veces que se itera la línea.
- iterate: Realiza la iteración retornando una lista de las instrucciones dentro segun el numero que se le haya especificado

- condWhile : Si la condición es verdadera se ejecuta lo de adentro, y al acabar vuelve al Begin del inicio del while
- cond_if_check: Revisa si es un if, si lo es, retorna un número que es la condición según un diccionario llamado condiciones
-
- End: Revisa si es un en, si lo que se retorna verdadero, retira un valor a la lista BEGIN
- acción: Revisa que acción se está ejecutando, y las agrega a una lista
- Cond: Revisa qué condición se está haciendo, retorna true si la condición se cumple
- NewBeepr(): Agrega a una lista, la posición actual de cada beeper en cada paso

3. Main_Execution.py: Este archivo comienza definiendo variables como el tamaño de la ventana, booleanos para el uso de botones y rectángulos para las interacciones mediante funciones como:
    - boton: Permite crear botones a partir de un Rect, una fuente y una variable booleana, al cambiar el valor booleano del botón en la siguiente iteración del while principal se entra en una condición que muestra contenido diferente
    - titulos: Permite insertar una línea de texto con tamaño grande
    - texto: Permite insertar texto en un Rect centrando
    - textosmall: Permite insertar texto pequeño en un Rect centrando
    - creditos: Inserta los créditos de los creadores
    - texter: Permite renderizar párrafos de texto sobre la ventana de pygame a partir de una subfunción llamada texte que divide los párrafo en líneas de cierta cantidad de caracteres

Se usaron condicionales para definir lo que se muestra en pantalla a partir del valor booleano dado a los botones. Además se usaron las funciones principales de pygame para insertar imágenes, dar color al fondo o definir la ventana. El módulo “sys” se usó para poder terminar la ejecución cerrando la ventana. mediante el módulo “os” nos fue posible ejecutar el archivo “ejecucionview” dentro del Main_Execution, de manera que se muestran paralelamente e interactúan

Video Py.Karel:
https://drive.google.com/drive/folders/1RKeNHLrW2HFweWNgslTKsfuBM-tfmWVy