Universidad Nacional Experimental Simón Bolívar
Ingeniería en Computación
CI2691: Laboratorio de Algoritmos y Estructuras I

La Vieja Cool

Profesor:
Fernando Lovera

Integrantes:
Jennifer Gámez 16-10396
Amaranta Villegas 16-11247

11 de Junio del 2019

Introducción

La programación es un proceso que se utiliza para idear y ordenar las acciones que se realizan en el marco de un proyecto.

En la actualidad la noción de programación se encuentra muy asociada a la creación de videojuegos y aplicaciones informáticas (Laboada, 1985).

En este estudio se plantea diseñar un proyecto de programación que permita jugar el tradicional juego de la vieja conocido por todos, incrementando su nivel de complejidad a través de una serie de modificaciones para así demostrar los conocimientos adquiridos en el desarrollo de la asignatura (CI2691 Laboratorio de Algoritmo y Estructuras).

Para desarrollar el juego se utilizará el lenguaje de programación PYTHON el cual utiliza una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional y para la parte de la interfaz gráfica utilizaremos las librerías de Tkinter y PyGame. La primera es muy utilizada la interfaz gráfica de usuario para Python y la última permite la creación de videojuegos en dos dimensiones de una manera sencilla(Sweigart, 2008).

Motivación

El deseo de medir nuestra capacidad de poder desarrollar un juego utilizando todos los conocimientos aprendidos en la materia.

Objetivos planteados

-Elaborar la lógica del juego de la vieja tridimensional a través del lenguaje de programación Python y el editor de texto sublime text con la finalidad de obtener un código que muestre un resultado final del juego.

-Diseñar una interfaz gráfica de usuario que permita la interacción entre el usuario y el videojuego.

Alcance de la solución

Contenido del informe

En el informe se reporta la estructura del trabajo a desarrollar, comenzando por el diseño en el que se presenta la manera de cómo se abordó el problema y las funciones y procedimientos a implementar, los detalles de implementación que nos permiten saber qué tipo de datos utilizamos y la estructuración del código también se comentará la operatividad del programa y las posibles anomalías que puede

presentar y para cerrar se reportarán los resultados obtenidos, las experiencias adquiridas ,las dificultades adquiridas y las recomendaciones.

DISEÑO

-Resultados del análisis descendente

En base al análisis descendente se inició el desarrollo del problema a partir de la elaboración de un tablero N x N, porque es necesario tener la misma cantidad de columnas que de filas para poder hacer las jugadas en el tablero.

Luego se creó un subprograma en el que se desarrolló un multitablero que almacena todos los tableros en el que se puede jugar. Después se creó una función en consola el tablero a trabajar y finalmente se desarrollaron las funciones que conforman una serie de pasos para realizar las jugadas en tablero.

A continuación se describen los subprogramas utilizados:

Dimensión. Esta es una función de entrada y es la que retorna dimensión que se quiere jugar y verifica si la funciones es válida, es decir, una dimensión mayor o igual que 2.

Función. Es una función de entrada pide como unicas entradas la identificación de los jugadores la misma verifica que este espacio este lleno de al menos un caracter.

Tablero_NxN. Se utilizará para almacenar los elementos a rellenar.

Super_tablero. Se creó un Array que almacenará el conjunto de tableros a utilizar.

MostrarTableros: Esta función permite mostrar en la consola el tablero por filas.

Jugada_Valida. Esta función determina si una jugada está dentro de los parámetros del tablero y si no está ocupada.

QueQuieresSer: Esta función permite preguntar al jugador si desea tener como fichas la “X” o la “O”.

QuienIniciapartida: Esta función permite decidir si comienza la partida el jugador 1 o el jugador 2.

JugadaTableros: Le permite al jugador decidir en qué posición colocará su ficha. Esto incluye la entrada de selección de una fila y columna.

ChooseTablero: Permite al jugador la elección del tablero en el que quiere jugar y verifica si la referencia es válida.

OkTablero: Aquí se comprueba que el tablero seleccionado es el deseado por el usuario. Rectifica que las información es la deseada por el usuario.

HayAlinea. Esta función permite determinar la existencia de líneas entre tableros o en un tablero específico. Cuando nos referimos a la existencia de líneas hablamos de que se encuentren fichas del mismo jugador consecutivas en un mismo plano o que se encuentren fichas del mismo jugador ocupando la misma casilla en los N planos jugados. El problema de determinar si hay línea se dividió en cuatro subproblemas:

El subproblema 1. (HayAlineaHorizontal). Determina si existe una línea horizontal en un tablero.

El subprograma 2 (HayAlíneaVertical). Determina si existe una línea vertical en un tablero.

El subprograma 3. (HayAlíneaDiagonal). Determina si existe línea diagonal en un tablero.

El subprograma 4.(HayAlíneaTableros). Determina si existe la misma ficha en la misma posición de cada tablero del juego.

Sumarlineas. Evalúa cada vez que un jugador tiene una línea y la va guardando en una variable que aumenta.

Otra Partida. Este procedimiento contiene al programa principal dentro de él, luego que se verifica que jugador obtuvo más líneas y se elige el ganador. Se pregunta si se desea jugar otra partida y de ser afirmativo llamamos a este procedimiento para que se vuelva a iniciar una nueva partida.

Demostraciones de correctitud

Subproblema de verificar si una jugada es válida

5.1 Según la descripción del problema, ¿bajo qué condiciones una jugada es válida?

Se considera que una jugada es válida si la casilla seleccionada por un usuario entra en los parámetros de la dimensión en la cual se está trabajando, o si esta casilla no se encuentra ocupada por una ficha; casilla vacía. No es válido la transcripción de una casilla más de una vez, y no es válido seleccionar un tablero inexistente o casilla.

5.2 Proponga una especificación de una función EsValida, con la siguiente forma:

booleano func EsValida

[

(int T : M, N: integer, int T: array [0..N)x[0..M) of integer; int: Fila, Columna: integer; out Valido: bool)

{ Pre: (columna < M) /\(columna >= 0) /\(fila < N) /\ (fila > 0)}

{ Post: T[fila][columna] == 0 /\ Valido == True) V (T[fila][columna] =/= 0 == True /\ Valido == False)}

Valido := False

if T[fila][columna] == 0 -> Valido:= True

[] T[fila][columna] =/= 0 -> Valido:= False:

fi

>> Valido

]

5.3 Escriba las instrucciones de la función EsValida.

1. Es Valido := True; cuando la casilla esta vacia.

2. Es Valido := True; cuando la casilla existe en las dimensiones del tablero en la cual se trabaja.

5.4 Demuestre la correctitud de la función EsValida.

Prueba 1:

(columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) => T[fila][columna] == 0 \/ T[fila][columna] =/= 0

(T[fila][columna] == 0 \/ T[fila][columna] =/= 0)

= < P V - P == True >

True

= < P V True == True >

= < H0 == True >

(columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0)

Prueba 2:

{ (columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] == 0 } Valido:= True {T[fila][columna] == 0 /\ Valido == True) V (T[fila][columna] =/= 0 == True /\ Valido == False)}

(T[fila][columna] == 0 /\ Valido == True) V (T[fila][columna] =/= 0 == True /\ Valido == False) [Valido:= True]

= <Sustitucion Textual>

(T[fila][columna] == 0 /\ True == True) V (T[fila][columna] =/= 0 == True /\ True == False)

= < True = P = P >

= < P /\ True == P>

(T[fila][columna] == 0) V (T[fila][columna] =/= 0 == True /\ True == False)

= < True = P = P >

= < P /\ False == False >

(T[fila][columna] == 0) v False

= < P V False == P >

T[fila][columna] == 0

= < P /\ True == P >

(T[fila][columna] == 0) /\ True

= < True == (columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] == 0 >

(T[fila][columna] == 0) /\ ((columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] == 0)

=> < p /\ q => p >

(columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] == 0

Prueba 3:

{ (columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] =/= 0 } Valido:= False {T[fila][columna] == 0 /\ Valido == True) V (T[fila][columna] =/= 0 == True /\ Valido == False)}

(T[fila][columna] == 0 /\ Valido == True) V (T[fila][columna] =/= 0 == True /\ Valido == False) [Valido:= False]

= <Sustitucion Textual>

(T[fila][columna] == 0 /\ False == True) V (T[fila][columna] =/= 0 == True /\ False == False)

= < True = P = P >

= < P /\ True == P>

(T[fila][columna] == 0 /\ False == True) V (T[fila][columna] =/= 0)

= < True = P = P >

= < P /\ False == False >

False V (T[fila][columna] =/= 0)

= < P V False == P >

T[fila][columna] =/= 0

= < P /\ True == P >

T[fila][columna] =/= 0 /\ True

= < True == (columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] == 0 >

T[fila][columna] =/= 0 /\ (columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] == 0

=> < p /\ q => p >

(columna < M) /\ (columna >= 0) /\(fila < N) /\ (fila > 0) /\ T[fila][columna] == 0

6.1. Demuestre la correctitud de la función HayAlineaHorizontal.

booleano func HayAlineaHorizontal (int N,M: integer, int T: array [0..N)x[0..M) of integer, int Columna, Ficha: integer, out HaylineaVertical: bool)

{ Pre: (columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O") }

{ Post: (i forall: 0 <= i < N /\ j == Columna: T[i][j] == Ficha) V (i exist: 0 <= i < N /\ j == Columna: T[i][j] =/= Ficha) /\ HayLineVertical}

HayLineaHorizonal := True

j := columna # valor fijo porque la columna sera la misma a verficar en casa fila

{inv:(i forall: 0 <= i < N /\ j == Columna: T[i][j] == Ficha) V (i exist: 0 <= i < N /\ j == Columna: T[i][j] =/= Ficha)}

{cota t: N-i}

do i < N /\ HayLineaHorizonal:

    if T[i][j] == Ficha -> Skip

    [] T[i][j] =/= Ficha -> HayLineaHorizontal:= False

    fi

od

>> HayLineaHorizonal

Pruebas de Correctitud

Prueba 1:

(columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O") -> T[i][j] == Ficha V T[i][j] =/= Ficha

T[i][j] == Ficha V T[i][j] =/= Ficha

= < P V -P == True >

= < True == P = P >

True

= < P V True == True >

True V (columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O")

<= < P <= P V Q >

(columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O")

Prueba 2:

{(columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O") /\ T[i][j] == Ficha } Skip {(i forall: 0 <= i < N /\ j == Columna: T[i][j] == Ficha) V (i exist: 0 <= i < N /\ j == Columna: T[i][j] =/= Ficha) /\ HayLineVertical }

(i forall: 0 <= i < N /\ j == Columna: T[i][j] == Ficha) V (i exist: 0 <= i < N /\ j == Columna: T[i][j] =/= Ficha /\ HayLineVertical)

= < p v -p == true >

True

= < P V True == True >

True V (columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O")

<= < P <= P V Q >

(columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O")

Prueba 3:

{(columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O" /\ T[i][j] =/= Ficha } HayLineaHorizontal:= False { ((i forall: 0 <= i < N /\ j == Columna: T[i][j] == Ficha) V (i exist: 0 <= i < N /\ j == Columna: T[i][j] =/= Ficha)) /\ HayLineVertical }

((i forall: 0 <= i < N /\ j == Columna: T[i][j] == Ficha) V (i exist: 0 <= i < N /\ j == Columna: T[i][j] =/= Ficha)) /\ HayLineVertical [HayLineaHorizontal:= False]

= < sustitucion textual >

((i forall: 0 <= i < N /\ j == Columna: T[i][j] == Ficha) V (i exist: 0 <= i < N /\ j == Columna: T[i][j] =/= Ficha)) /\ False

= < p /\ False == False >

False

<= < p /\ q => p , p/\ true== true >

true

= < true == (columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O" /\ T[i][j] =/= Ficha >

(columna < M) /\ (columna >= 0) /\ (fila < N) /\ (fila > 0) /\ (Ficha == "X" V Ficha == "O" /\ T[i][j] =/= Ficha

Optimización

Se realizaron distintas modificaciones a la parte lógica del programa para que tuviera un funcionamiento adecuado.

DETALLES DE IMPLEMENTACIÓN

Tipo de datos.

En Python todos sus elementos son objetos y los datos una vez identificados se convierten en objetos instanciados del tipo al que pertenecen.

Se trabajó con varios tipos de datos como son: los números enteros (int) que sólo interpreta números enteros.

String. Se utiliza para guardar y representar texto y se utilizó para guardar datos como fueron: los nombres de los jugadores y las fichas de los jugadores.

Tipos Booleanos. Son una especie de tipos numéricos para evaluar si una expresión lógica es cierta (True) o es falsa (false). Se utilizó para verificar si las jugada son válidas, para verificar si en det choose tablero la referencia del tablero se encontraba en el parámetro, en hay líneas se asignaron variables booleanas que permitían saber si existía línea o no.

Listas. Una lista es una secuencia ordenada de elementos mutables. Se utilizan (matrices) para representar cada tablero y se utilizó un super arrays que contenía el conjunto de tableros llamados Def Supertablero.

Estructura del Código

El juego inicia con la llamada de la función de la dimensión en la que se quiere jugar teniendo por entrada el tamaño deseado, luego se hace el llamado de la funcion Identificacion y este procede preguntando a los jugadores sus nombres, más adelante se implementa la función que quiere ser, la cual permite asignar la ficha a ambos jugadores, después se implementa la función ¿Quién inicia partida? Que permite reafirmar la información proporcionada por los usuarios y decidir quien inicia la partida, después se trabaja con la función de supertablero la cual construye los tableros a partir de la dimensión seleccionada, luego se trabaja con la función, choose tablero que permite la confirmación del tablero deseado , luego se trabaja con la función OK tablero que indica si es el tablero deseado o no, luego se trabaja con la función jugada tablero que permite hacer la jugada de cada jugador, después se trabaja con la función HayAlinea cuya función permite verificar si los jugadores realizan línea o no, luego se trabajó en sumar línea para ir verificando cuantas líneas lleva cada jugador y por último se trabajó con el procedimiento otra partida que dependiendo si el usuario desea jugar nuevamente se realiza una nueva partida o se termina el juego.

Estado Actual

-Operatividad del programa (Lógica)

La parte lógica del programa en general funciona correctamente pero se nos hizo imposible el poder relacionar la interfaz gráfica con la parte lógica

Manual de operación:

Archivos a ejecutar: **ParteLogica.py** (Solo la parte lógica)

.- **Nota:** las funciones que fueron llamadas con valores iniciales, generalmente, dichos valores es donde la función lee el inconveniente y ejecuta la aserción robusta con el fin de permitir corregir la información suministrada, esto mayormente se hizo cuando las entradas eran del tipo entero para evitar interrumpir el código con string.

Modo de operar el programa:

El juego inicia con la llamada del procedimiento principal OtraPartida, este es un procedimiento compuesto de varias funciones por análisis decentes que permiten una mayor distribución del código en cuanto a procesar datos, almacenar y asignar. Se hace la llamada Funcion Dimension cuyo objetivo es de permitir una entrada numérica mayor o igual a 2; este dato se almacena en DimensionesTab. Posterior se muestra la salida de un mensaje con las dimensiones seleccionadas, se hace la llamada de la funcion Identificacion es aqui donde los usuarios registran sus nombres o cualquier otro distintivo, y se asigna a variables constantes y con estas se hace el llamado de la próxima función QueQuiereSer para asignar fichas según el gusto de cada usuario; estos datos pueden ser corregidos y por ello se pide la confirmación de datos (assert por cada función). Con la constante proporcionada al comienzo del juego emplea para crear las dimensiones en un SuperTablero como le hemos asignado, este super tablero contiene todos los tableros de tamaños= DimensionesTab.

Se continua el código con la creación de la variable Ficha; este se comporta como un contador que se encuentra en constante empleo con el bucle, por cada interacción exitosa con los jugadores esta variable disminuye. Se inicializa las líneas por cada jugador y se entra en la interacción de la parte lógica con cada jugador dependiendo del turno. Se estableció que los turnos de cada jugador viene dado por unos condicionales el cual cada ficha par es el turno de Jugador 1 y cada ficha impar el Jugador 2. Se piden de entrada dentro de este bucle y con el primer condicional escoger un tablero de preferencia con la función llamada ChooseTablero, seguido de esto se ejecuta la función OkTablero y esto no es más que una función de procesamiento de datos que busca reafirma los datos proporcionados en cuanto al tablero de selección por el jugador de turno, si se reafirma la elección se hace la llamada de la función JugadaTablero y esta tiene por entrada una fila y columna en las dimensiones que se se seleccionó al principio de ejecutar el juego, si esta jugada es válida entonces se imprime en el tablero en caso contrario se llama nuevamente la función JugadaTablero. La

funcion JugadaTablero tiene la caracteristica de que esta retorna la fila y columna jugada y gracias a esto se hace el llamado de la función HayALinea, en caso de la existencia de linea se pone en funcionamiento la función SumarLinea almacenando la información en las líneas por jugadores antes establecida, cada fase por jugador es reflejada en la shell y al finalizar el turno de un jugador se refleja las líneas acumuladas por el momento. Todo lo anterior se repite con el segundo Jugador que entra en el condicional dos, y por medio del bucle este proceso es interactivo por lo que permite intercalar los turnos. Una vez acabada las Fichas se ejecuta un condicional para determinar cuál de los dos jugadores es el ganador (es posible determinar si es empate), le precede una entrada de volver a jugar o finalizar el juego.

CONCLUSIONES

Resultados obtenidos

Se logró diseñar el juego de manera que se puede desarrollar perfectamente en una terminal sin el uso de una interfaz para la aplicación del mismo; se logra el objetivo de interacción y modificación de datos a lo largo del juego, esto en cuanto a parte lógica se refiere y en cuanto a la segunda parte del problema: interfaz gráfica, se pudo diseñar una ventana en la cual los jugadores ingresarán sus nombres y la dimensión de tablero a jugar, luego de eso se despliega un tablero

con la entrada proporcionada, sin embargo esta no tiene utilidad alguna debido a que no logramos relacionar la parte lógica con la parte gráfica por lo que no hay una interacción con los usuarios después de las primeras entradas.

Experiencias adquiridas

.- Este proyecto nos permitió poder detectar con más rapidez algunos errores comunes que se presentan al momento de programar en Python.

.-Aprendimos como diseñar un juego en el lenguaje Python, en este caso, el juego de la vieja con algunas modificaciones.

.-Adquirimos más seguridad y pudimos conocer más sobre la sintaxis del lenguaje y como escribir más funciones recursivas y poder implementarlas.

.-Aprendimos a ser autodidactas en el manejo del lenguaje y en la búsqueda de herramientas que se pueden utilizar para obtener la solución de un problema.

Dificultades presentadas

.- La situación actual del país fue una limitante en muchos aspectos, tales como, los cortes de luz sin previo aviso nos limitaron en el tiempo de dedicación del proyecto y en lo particular (Amaranta Villegas) el cual presentó inconveniente con su equipo electrico de trabajo: quema de la tarjeta madre de la computadora.

.-La falta de transporte impidió el encuentro regular del equipo de trabajo para poder desarrollar las actividades planificadas. Así como también impidió recibir las orientaciones físicamente con el docente.

.-El total desconocimiento de las librerías de interfaz gráfica de Python, tales como, Tkinter y Pygame que son estructuras necesarias para poder desarrollar una interfaz gráfica a un programa.

.-Al ser los tutoriales de Tkinter y Pygame encontrados en internet, muy básicos no permitieron obtener la información necesaria o suficiente para el desarrollo de la interfaz gráfica que se necesitaba.

Recomendaciones:

.-Suministrar a los futuros estudiantes mayor información sobre las librerías de interfaz gráfica.

Referencias Bibliográficas

Becerril, Jesús. [Sprogramación]. (2014,septiembre,01).Python 3.X Tkinter Tutorial 4 entradas (Archivo de videos). Recuperado de https://www.youtube.com/watch?v=lPxIcYin8y0&list=PLdLhegjjdnsBbWIjHXmlYv0AJ85q1SMBt&index=4

Laboada, Xavier; Josep,Yolimary; Pena, Rosa María y Gual Antoni. (1985).”Software”. Biblioteca práctica de la computación. Barcelona: Ediciones Océano-Éxito,S.A.

Https://pythonista.io/cursos/py101/tipo-de-datos-y-operadores.

[Píldoras Informáticas]. (2018, enero,31).Curso Python. Interfaces gráficas I. Video 42. (Archivo de video). https://www.youtube.com/watch?v=hTUJC8HsC2I&t=17s

Python Software Fundatión. Tkinter-Python interface to Tel/Tk. Junio 1.2019. 2 https://docs.python.org/3.6/library/tkinter.html

Sweigart, A.(2008). Inventa tus propios juegos de computadoras con Python. Pág. Traducido por Carella, Alfredo; Pernión, Alejandro y Palm, Francisco. https://inventwithpython.com/es/InventarConPython_3a_es.pdf

Wikibooks.ORG. Python/Interfaz gráfica con Tkinter/los nombresde los colores. Junio 01-2019. https://es.wikibooks.org/wiki/Python/Interfaz_gr%C3%A1fica_con_Tkinter/Los_nombres_de_los_colores

.