Proyecto 1 - Blokker

201901973 – Raudy David Cabrera Contreras

Resumen

La realización de este proyecto conto con programación orientada a objetos, donde se implementaron tipos de datos abstractos tal como la matriz dispersa para poner crear un tablero de mxn donde se crean nodos a partir de las coordenadas que se muestran en la interfaz gráfica, donde allí se encuentran todas las funciones que posee este programa. Donde puede iniciar la partida creando un tablero, para luego ingresar el color de piezas que quiere para su partida, solo quedaría que cada jugador ingrese las coordenadas de donde quiere ubicar la pieza, que es una inserción de nodo en la matriz dispersa, donde la matriz dispersa tiene dos listas enlazadas utilizadas como listas de cabecera para recorrer de una forma más fácil las casillas donde se insertaran las piezas del juego, usando como id el color de cada jugador, donde puede ser cuatro colores para los jugadores, donde no se puede repetir el color del otro.

Palabras clave

Matriz dispersa

Tkinter

Xml

Graphiz

Abstract

The realization of this project had object-oriented programming, where abstract data types such as the sparse matrix were implemented to create an mxn board where nodes are created from the coordinates shown in the graphical interface, where there you will find all the functions that this program has. Where you can start the game by creating a board, and then enter the color of pieces you want for your game, it would only be for each player to enter the coordinates of where he wants to locate the piece, which is a node insertion in the scattered matrix, where the sparse matrix has two linked lists used as header lists to go through in an easier way the squares where the game pieces will be inserted, using as ID the color of each player, where it can be four colors for the players, where it cannot be repeat the color of the other one.

Keywords

Matrix scatter

Tkinter

Xml

Graphiz

Introducción

El desarrollo de este proyecto de programación fue programado en el lenguaje de Python, donde se importaron varias librerías, tanto como para la lectura de archivos, mostrar por medio de interfaz grafica las funciones del proyecto. El manejo de memoria fue dinámico, donde se utilizaron tipos de datos abstractos (TDA), para el manejo de nodos en forma de tablero.

El software realizado es el de un juego, clásico tipo Tetris, donde se colocan coordenadas para insertar una pieza en el tablero.

Desarrollo del tema

Para el desarrollo de este proyecto con el lenguaje de programación Python, en primer lugar, se importaron librerías que nos ayudaran con algunas funcionalidades para el juego, las librerías utilizadas son:

TKinter: utilizada para la interfaz gráfica.

MiniDom: para la lectura de los archivos XML

Os: módulo provee una manera versátil de usar funcionalidades dependientes del sistema operativo.

Time: modulo que para llevar el tiempo de los jugadores

Modo de Juego:

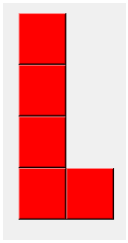
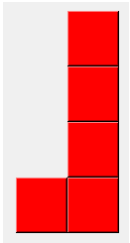
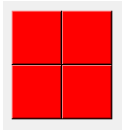
Se creará un tablero de MxN, donde se ubicarán las piezas insertadas por los jugadores, el juego empezara cuando se dé el botón de crear tablero del juego. Únicamente podrán jugar dos personas por partida. Una vez dadas las dimensiones del tablero, colocara el color de pieza que desea cada jugador. Los jugadores no pueden escoger el mismo color.

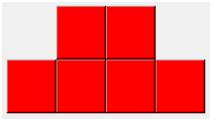
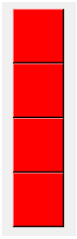
Un jugador obtendrá un punto por cada vez que logre insertar una pieza en el tablero. Cada jugador dispondrá de 1 minuto de juego. El juego finaliza cuando ya no se encuentren disponibles los espacios para colocar una pieza en el tablero.

Piezas:

Las piezas saldrán de forma aleatoria cuando inicie la partida. Se podrá visualizar la siguiente pieza que se insertará en la posición que el jugador decida, poniendo el numero de la fila y el numero de la columna donde quiera ubicar la pieza que se visualiza a un lado del tablero.

Las piezas que trae el juego son las siguientes:

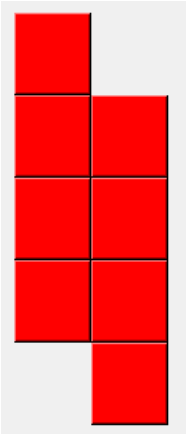
- 
- 
- 
- 

- 
- 

Restricciones de piezas:

Cuando se quiera insertar una pieza debe verificar que quepa en el tablero, que no se salga de los limites que se hayan puesto, ya que, al salirse de la pantalla, no lo colocará y no obtendrá puntos.

Las piezas de un jugador no pueden tocar ninguno de sus lados, solo pueden tocarse las esquinas. Si un jugador pone dos piezas pegadas o encima, pierde un punto y cambiara de turno del jugador.



Ejemplo de mal posicionado de una pieza.

Memoria:

La forma de guardar cada pieza es en un nodo que se inserta en una matriz dispersa, donde esta se utilizo ya que crea nodos cabecera dependiendo del tamaño que el usuario escoja, y en ese TDA en donde los nodos que se van insertando y guardando, para cuando se quiera imprimir por medio de Graphiz o en el reporte HTML.

Cuando se juega la partida, habrá una opción de guardarla, donde la matriz dispersa se transcribirá a XML, donde se guardará como jugador 1 y jugador 2, y forma se guardará en la matriz dispersa el color del jugador como id, de igual forma habrá una opción de cargar partida, que por medio de MiniDom se leerá y se insertará en el tablero y en la matriz dispersa.

Reportes:

Para mostrar la información de la matriz dispersa, como quedaron insertados los nodos en la misma, y los puntajes con colores de los jugadores.

Se creo un HTML donde se reporta la información de los jugadores con sus colores y puntajes, junto con cuantos movimientos se realizaron en la partida. Se utilizo HTML y CSS para hacer más agradable la información del juego.

Se utilizo también Graphiz, para representar de una forma mejor la matriz dispersa en forma de tablero, mostrando los nodos del color que los jugadores lo indicaron. Donde con la librería de Os, se abrirá al presionar la opción. Es una herramienta muy útil para transformar y abrir los archivos de forma eficaz.

Matriz Dispersa:

es un matriz en el que la mayoría de los elementos son cero. No existe una definición estricta de cuántos elementos deben ser cero para que se considere una matriz. escaso pero un criterio común es que el

número de elementos distintos de cero sea aproximadamente el número de filas o columnas. Por el contrario, si la mayoría de los elementos son distintos de cero, entonces la matriz se considera denso.[2]

TDA:

Un Tipo de Dato Abstracto (TDA) es un modelo que define valores y las operaciones que se pueden realizan sobre ellos. Y se denomina abstracto ya que la intención es que quien lo utiliza, no necesita conocer los detalles de la representación interna o bien el cómo están implementadas las operaciones.[1]

Conclusiones

Una de las ventajas de utilizar una matriz dispersa, es que cuando se crea no tiene nodos que conecten las dos listas enlazadas como cabeceras, eso hace que cuando se crean nodos y se quiera recorrer, no de mucha vuelta a la memoria ya que solo existe lo que se va creando, el demás espacio está nulos, por lo cual el recorrido es mucho más fácil.

Hacer la interfaz grafica fue un reto, ya que normalmente se utiliza Python y mostrando las cosas por consola. Se intento usar la librería e Pyqt5, pero era algo confuso así que al final el proyecto se realizo con la ayuda de la librería de Tkinter, donde su documentación es muy buena, y la forma de crear y controlar las funciones de los objetos de la interfaz fue mucho más fácil.

Referencias bibliográficas

1. Programación II - Tipo de Dato Abstracto - Google Sites

https://sites.google.com/site/programacioniiuno/temario/unidad-2---tipo-abstracto-de-dato/tipo-de-dato-abstracto

2. Vvikipedla: Matriz Dispersa
https://es.vvikipedla.com/wiki/Sparse_matrix

Anexos

Clase Nodo, es el nodo de la pieza que se inserta, ya que al estar en el tablero tiene un apuntador para los 4 puntos cardinales.

```
class Nodo: #TDA NODO, lo que esta como pieza
    def __init__(self, fila, columna, valor):
        self.valor = valor

        self.fila = fila
        self.columna = columna
        self.derecha = None
        self.izquierda = None
        self.arriba = None
        self.abajo = None
```

Clase nodoEncabezado, es el nodo que se utiliza como los ejes x, y.

```
class nodoEncabezado: #NODO de encabezado
    def __init__(self,id):
        self.id = id
        self.siguiente = None
        self.anterior = None
        self.accesoNodo = None
```

Clase Lista Encabezado, que fue utilizada para crear la lista doblemente enlazada utilizadas como cabecera a partir de la clase de NodoEncabezado.

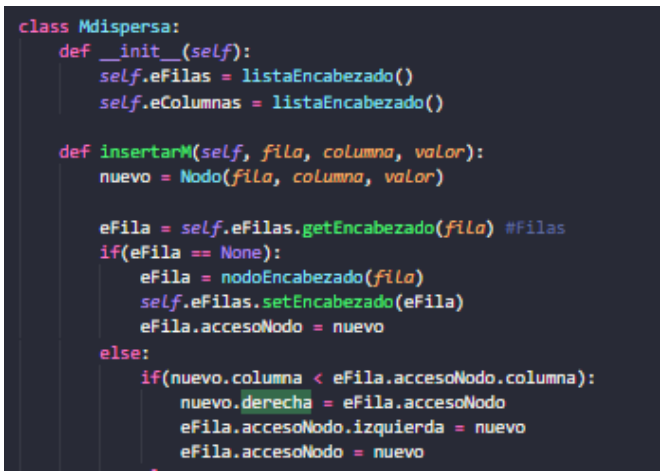
```
class listaEncabezado: #Lista doblemente enlazada
    def __init__(self, primero=None):
        self.primero = primero

    def setEncabezado(self, nuevo): #insertar nodo cabecera
        if self.primero == None:
            self.primero = nuevo
        elif nuevo.id < self.primero.id: #validar si el nuevo valor en menor que la cabeza primero
            nuevo.siguiente = self.primero
            self.primero.anterior = nuevo
            self.primero = nuevo
        else:
            actual = self.primero #para recorrer
            while actual.siguiente != None: #insertar en medio, 2
                if nuevo.id < actual.siguiente.id:
                    nuevo.siguiente = actual.siguiente
                    actual.siguiente.anterior = nuevo
                    nuevo.anterior = actual
                    actual.siguiente = nuevo
                    break
                actual = actual.siguiente

            if actual.siguiente == None: #insertar al final nodo 4
                actual.siguiente = nuevo
                nuevo.anterior = actual

    def getEncabezado(self,id): #como busqeuda
        actual = self.primero
        while actual != None:
            if actual.id == id:
                return actual
            actual = actual.siguiente
        return None
```

Clase Mdispersa, que fue utilizada para crear el tablero del juego, donde se insertan los nodos de la clase Nodo



```
class Mdispersa:
    def __init__(self):
        self.eFilas = listaEncabezado()
        self.eColumnas = listaEncabezado()

    def insertarM(self, fila, columna, valor):
        nuevo = Nodo(fila, columna, valor)

        eFila = self.eFilas.getEncabezado(fila) #Filas
        if(eFila == None):
            eFila = nodoEncabezado(fila)
            self.eFilas.setEncabezado(eFila)
            eFila.accesoNodo = nuevo
        else:
            if(nuevo.columna < eFila.accesoNodo.columna):
                nuevo.derecha = eFila.accesoNodo
                eFila.accesoNodo.izquierda = nuevo
                eFila.accesoNodo = nuevo
```

Repositorio de GitHub:

https://github.com/201901973-RaudyCabrera/IPC2_Proyecto1_201901973_Junio2021