Análisis de Algoritmos - Proyecto de Flow Free mediante algoritmo automatizado con heurística

Pablo Santander Álvarez

Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
santander_pjaveriana.edu.co

23 de noviembre de 2022

Resumen

En este documento se presenta la formalización e implementación de un algoritmo construido respecto al juego de Flow Free que tiene el fin de jugar de manera automática un tablero dado, donde mediante heurística y backtracking intenta completar un nivel conectando todos los puntos con su color respectivo. En caso de que el algoritmo no logre solucionar el tablero, se indica también que el algoritmo "no lo logro". Adicionalmente, se presenta una sección de resultados obtenidos al probar el algoritmo con una serie de 4 tableros.

Palabras clave: programación, python, backtracking, flow free, algoritmo, formalización, experimentación, complejidad, heurística, aprendizaje de maquina.

Índice

1. Introducción

El presente documento tiene como objetivo mostrar el proceso realizado para construir un algoritmo que juegue Flow Free de manera automática mediante el uso de heurística con la Distancia de Manhattan (también conocida como distancia de taxi) esta es una medida de la distancia entre dos puntos en un plano 2D, donde la ruta entre estos dos puntos tiene que seguir el diseño de la cuadrícula o en este caso, del tablero generado. Se basa en la idea de que un taxi tendrá que permanecer en la carretera y no podrá atravesar edificios. Se pretende mostrar: la formalización del problema (sección 3), la escritura formal del algoritmo (sección 4) y una sección de resultados obtenidos.

2. Librerías usadas en Python

Turtle	Turtle es una biblioteca de Python preinstalada que permite crear imágenes y formas.
Time	La biblioteca time contiene una serie de funciones relacionadas con la medición del tiempo.

La razón de uso de Turtle, es que si bien esta librería fue diseñada para introducir la programación a personas nuevas y niños mediante el típico juego de tortuga a la que se le dan comandos para que se mueva; con la correcta configuración se pueden realizar cualquier tipo de representación 2D. Para esto caso se decidió usar Turtle para representar el tablero de Flow Free y además, poder evidenciar de manera visual como el algoritmo se va comportando y realizando los diferentes movimientos para poder lograr conectar todos los colores. En la interfaz visual se verán además, los colores y las lineas que cada uno representa.

3. Formalización del problema

Para el problema de un *Algoritmo que solucione tableros de Flow Free* inicialmente podemos partir de una matriz T determinado en sus dimensiones por $T : \mathbb{R}^{(n*m)}$ donde *nym* son dadas por el usuario mediante un menú inicial del algoritmo. Es importante resaltar que la primera verificación realizada es el hecho de que un color no puede estar rodeado completamente de otros colores, es decir, si en la esquina de la matriz esta el color rojo, y a su derecha y abajo hay otros colores, se determinaría inmediatamente que es un tablero invalido ya que desde su construcción inicial no es posible de solucionar.

3.1. Definición del problema de Algoritmo automático de Flow Free

Así, el problema se define a partir de:

- Entradas:
  - La entrada se puede definir como: Una matriz T, donde $T \in \mathbb{R}^{(n*m)}$ donde n son la cantidad de filas de la matriz y m la cantidad de columnas.
- Salidas:
  - La salida es básicamente una impresión en Pantalla que indica que el Tablero ha sido resuelto. El proceso en el que el algoritmo construye diferentes caminos e intenta resolver el tablero es visual.

4. Algoritmo de solución

4.1. Descripción del Programa

Este algoritmo compuesto por diversas funciones adicionales que luego serán descritas, permite que un tablero de Flow Free sea resuelto de manera automática, donde a su vez se muestra de manera visual los caminos y decisiones tomadas por el algoritmo. De manera inicial, partimos de una matriz que no necesariamente debe ser cuadrada, donde sus dimensiones son determinadas bajo 4 distintas opciones de tablero que son desplegadas en un menú para el usuario. Al tener la matriz con sus puntos de origen de color establecidos, se realiza una identificación de todos los puntos o nodos que componen el tablero, con el fin de determinar en que posición están los nodos de origen y los nodos de fin, o aquellos con los que los nodos de origen deben conectar. Este proceso permite guardar los nodos en diccionarios con una llave que identifica el color. Una vez se logra esto, se procede a aplicar la Heurística descrita previamente en el documento la cual es la Distancia de Manhattan, gracias a un proceso de identificación que calcula la distancia de Manhattan respecto a todos los nodos y su distancia entre ellos, guardando en una lista ordenada los colores, iniciando por supuesto por el que menor distancia tiene.

En este punto, la parte visual realizada mediante Turtle entra en juego, donde se dibuja un tablero de un ancho determinado como 30, en este tablero mediante un pequeño algoritmo se añaden todos los nodos identificados con el fin de graficarlos. Finalmente se dibuja un cuadrado del tamaño del tablero. Es aquí donde se realiza la validación inicial que le permite al algoritmo determinar si el tablero generado es un

tablero valido, el algoritmo es capaz desde un inicio indicar si el tablero es jugable mas no necesariamente solucionable, injugable seria aquel tablero donde por ejemplo existe un color rodeado completamente de otros colores. Existe también una función que valida si el tablero ha sido completado o no. Si estas validaciones pasan, el algoritmo encargado de construir los caminos inicia, mediante un ciclo se elige el primer nodo de inicio y el primer nodo de fin, basados en el color de la lista ordenada de distancias de Manhattan, una vez elegidos se realizan validaciones con condicionales para determinar la prioridad heurística que se le otorgara a cada dirección. De manera recursiva el algoritmo revisa los posibles caminos en su nodo de inicio, y la heurística determina el mejor siguiente movimiento.

4.2. Descripción de Funciones utilizadas

- drawtablero():
  - La función se encarga de graficar un plano de dos dimensiones en donde el juego se podrá visualizar.
- cuadrado():
  - La función se encarga de dibujar un cuadrado dentro de la pantalla generada por la función drawtablero(), construyendo los cuatro lados del cuadrado.
- validartablero():
  - La función se encarga de realizar una validación inicial respecto al tablero, donde tiene la capacidad de determinar si el tablero es jugable, con el fin de descartar aquellos que no lo sean.
- resuelto():
  - La función se encarga de validar si el tablero ha sido resuelto o no, revisando que todos los colores estén conectados y el tablero no tenga espacios vacíos.
- resolverTablero():
  - Algoritmo principal encargado de jugar en el tablero de manera automática mediante heurística y recesión.

4.3. Algoritmo automático que juega Flow Free - Heurística con Backtracking

4.3.1. Análisis de complejidad

Para determinar la complejidad del algoritmo podemos identificar que el algoritmo posee un ciclo dentro del cual existe el llamado recursivo al algoritmo. Para evaluar la complejidad utilizamos las siguientes ecuaciones:

- T(n) = T(n-1) + T(n-2) + T(n-3) + ... + T(0).
- T(n-1) = T(n-2) + T(n-3) + ... + T(0) reemplazamos T(n-1) en la primera ecuación.
- T(n) = T(n-1) + T(n-1)

Respecto a la tercera ecuación, podemos desarrollarla:
$= 2 * T(n-1)$
$= 2 * 2 * T(n-2)$
$= 2^n T(n-n)$
$= 2^n T(0)$ // Consideramos T(0) = 1
$= 2^n$
De esta manera, la complejidad algorítmica es de $0(2^n)$

Algoritmo 1 Algoritmo Recursivo y Automatizado Flow Free.

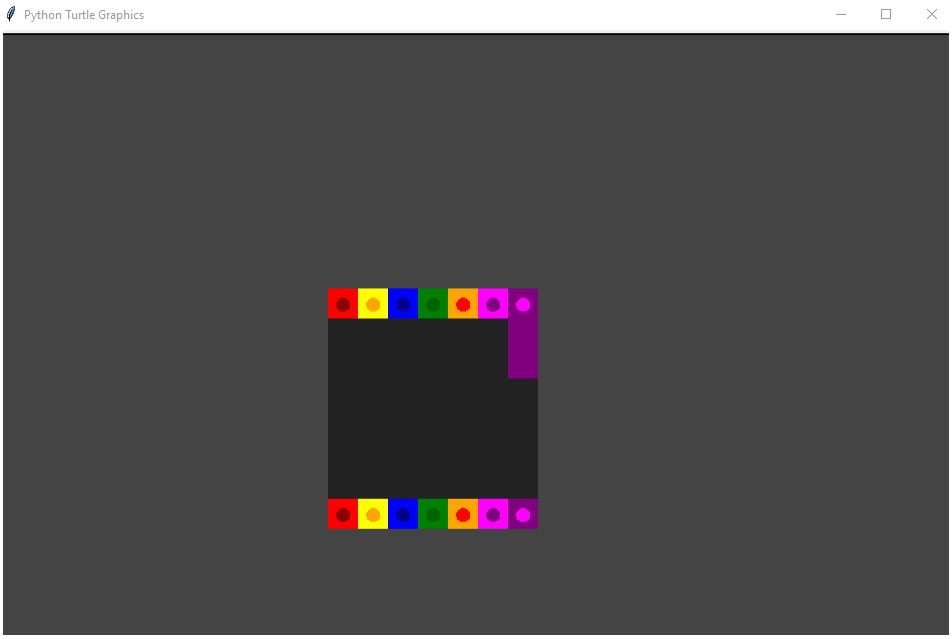
Require: $T \in \mathbb{R}^{(n*m)}$

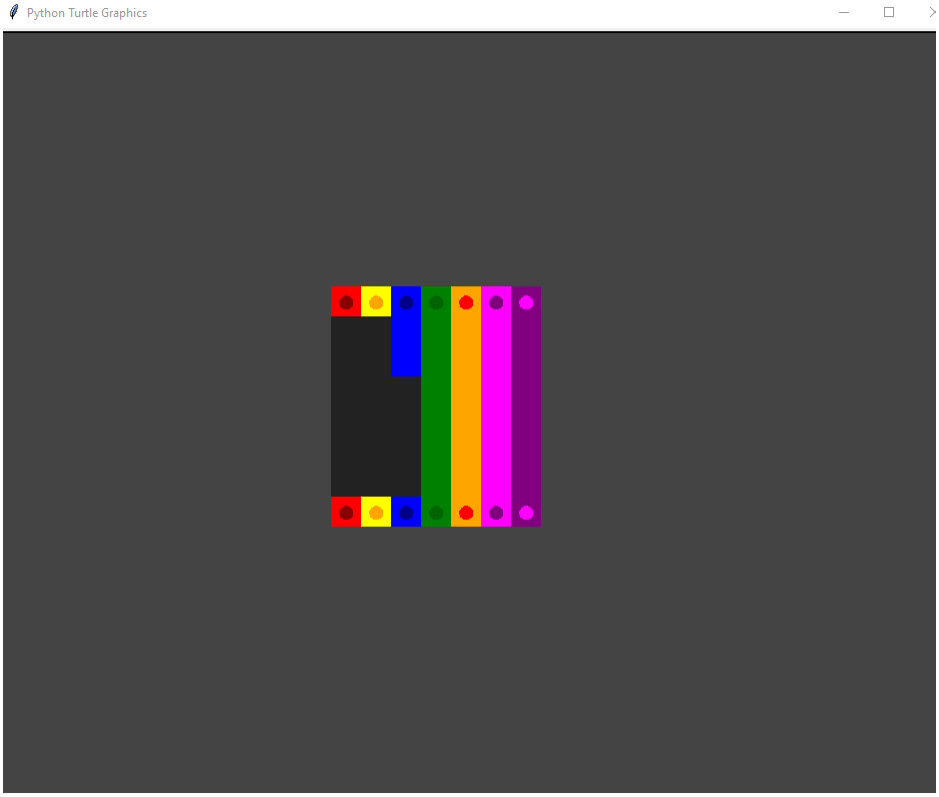
```
procedure RESOLVERTABLERO(T)
    DRAWTABLERO(30)
    turtle.getscreen().update()
    time.sleep(1 − VELOCIDAD)
    if VALIDARTABLERO(T)= False then
        return False
        if RESUELTO(T)= True then
            return True
            for color in nodosOrdenados do
                startNode ← nodosIniciales[color]
                endNode ← nodosFinales[color]
                if ABS(endNode[0], starNode[0]) + ABS(endNode[1], startNode[1])1 then
                    direcciones ← []
                    if tablero[startNode[0]][startNode[1] + 1] = 0 then
                        if endNode[1] > startNode[1] then
                            direcciones.insert(0, "derecha")
                        else
                            direcciones.insert(0, "derecha")
                        end if
                    end if
                    if tablero[startNode[0]][startNode[1] − 1] = 0 then
                        if endNode[1] < startNode[1] then
                            direcciones.insert(0, "izquierda")
                        else
                            direcciones.append(0, "izquierda")
                        end if
                    end if
                    if tablero[startNode[0] + 1][startNode[1]] = 0 then
                        if endNode[0] > startNode[0] then
                            direcciones.insert(0, "abajo")
                        else
                            direcciones.append(0, "abajo")
                        end if
                    end if
                    if tablero[startNode[0] − 1][startNode[1]] = 0 then
                        if endNode[0] > startNode[0] then
                            direcciones.insert(0, "arriba")
                        else
                            direcciones.append(0, "arriba")
                        end if
                    end if
                    if |direcciones| = 0 then
                        return False
                    end if
                    for direccion in direcciones do
                        if direccion = "derecha" then
                            startNode[1] = startNode[1] + 1
                            tablero[startNode[0]][startNode[1]] = color
                            if RESOLVERTABLERO(T)= True then
                                return True
                            else
                                tablero[startNode[0]][startNode[1]] = 0
                                startNode[1] = startNode[1] − 1
                            end if
                        end if
                        if direccion = "izquierda" then
                            startNode[1] = startNode[1] − 1
                            tablero[startNode[0]][startNode[1]] = color
```

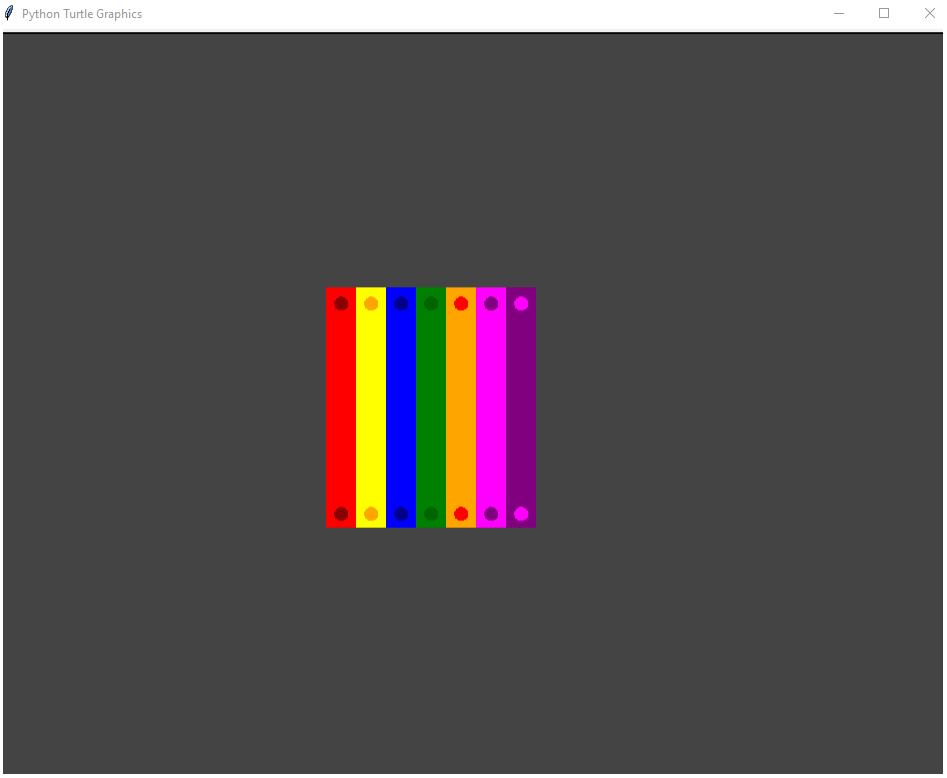
5. Resultados Obtenidos

En esta sección se presentan los resultados que han sido obtenidos mediante pruebas con diferentes tableros, donde se observo el comportamiento del algoritmo y la construcción recurrente de los caminos hasta finalizar la completitud del tablero de juego.

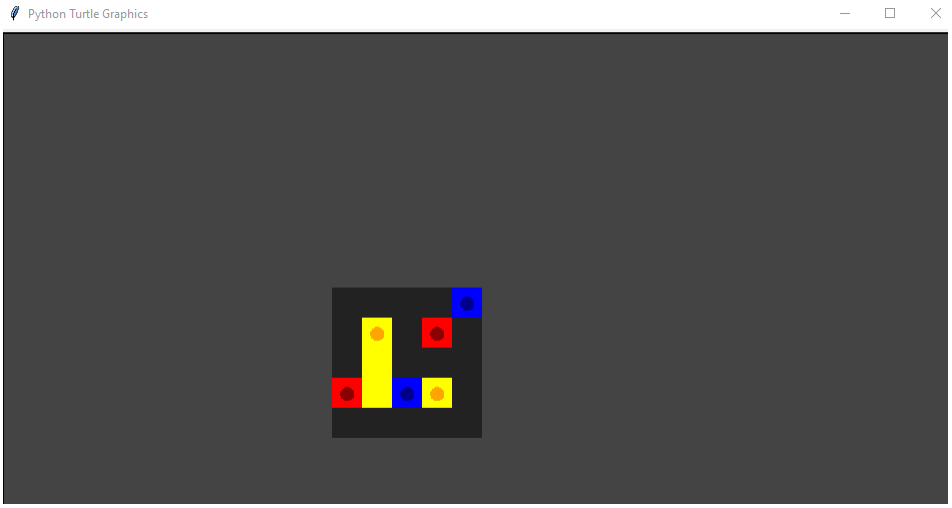
Resultado 1, Tablero 8x7

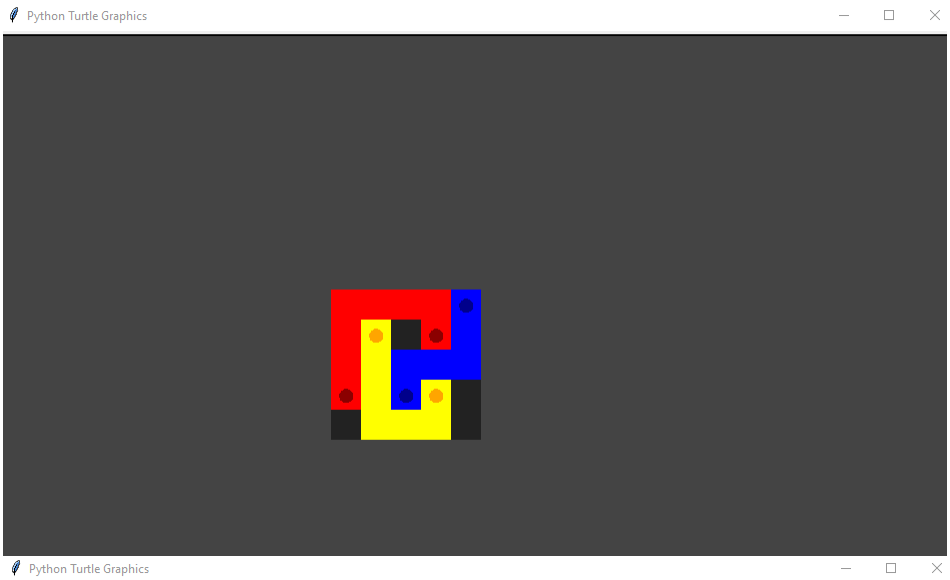


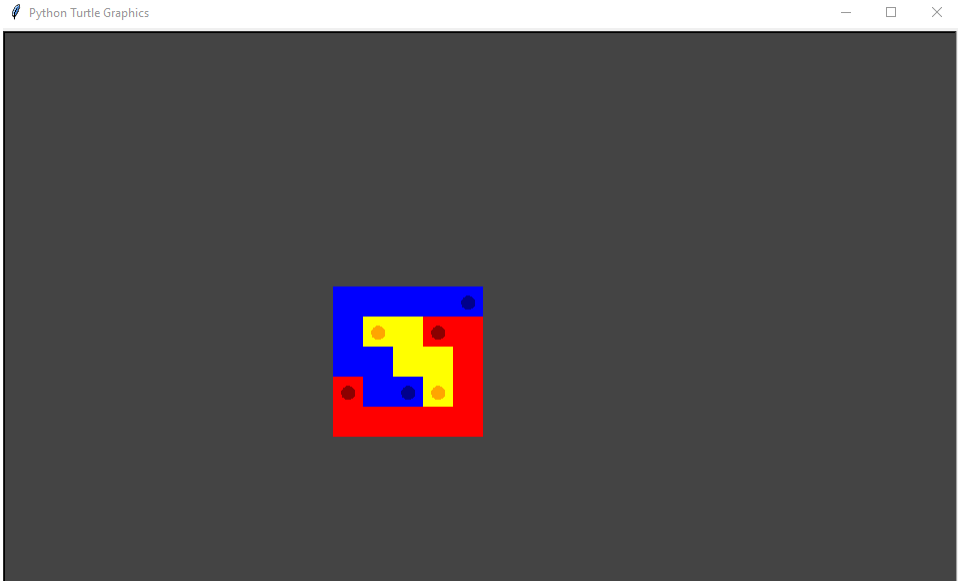




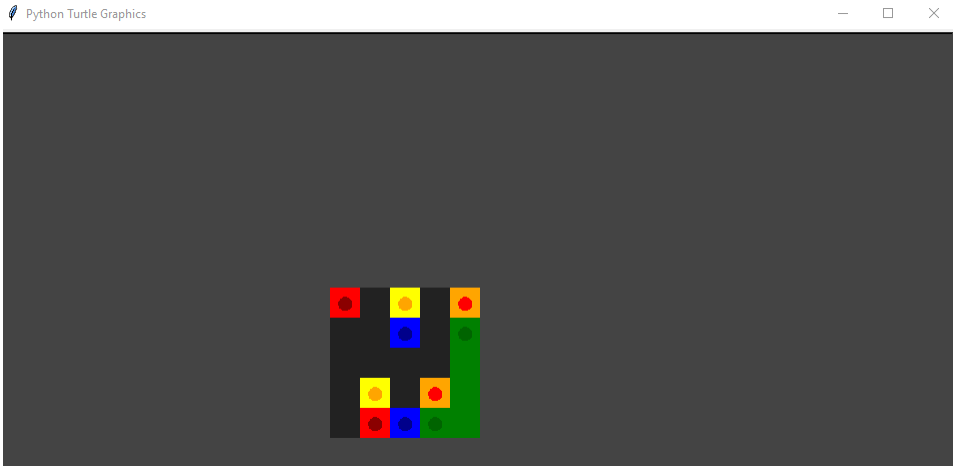
Resultado 2, Tablero 5x5

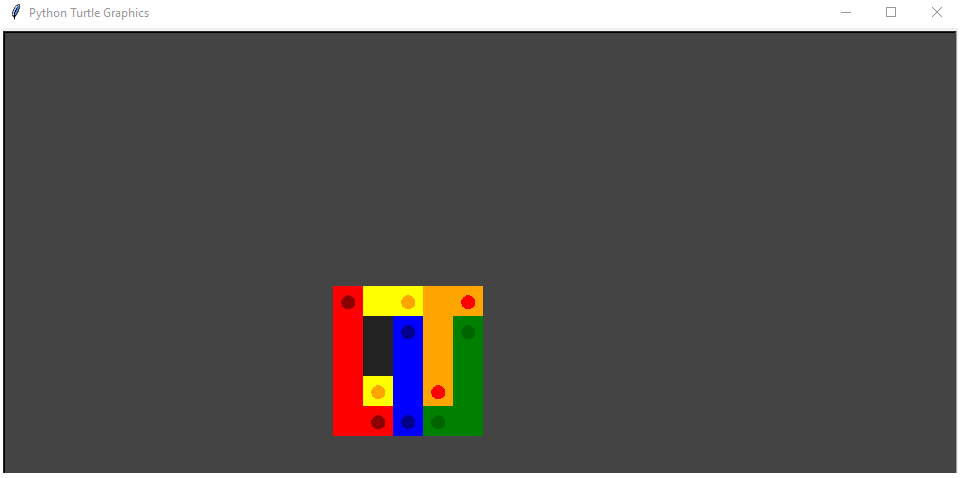


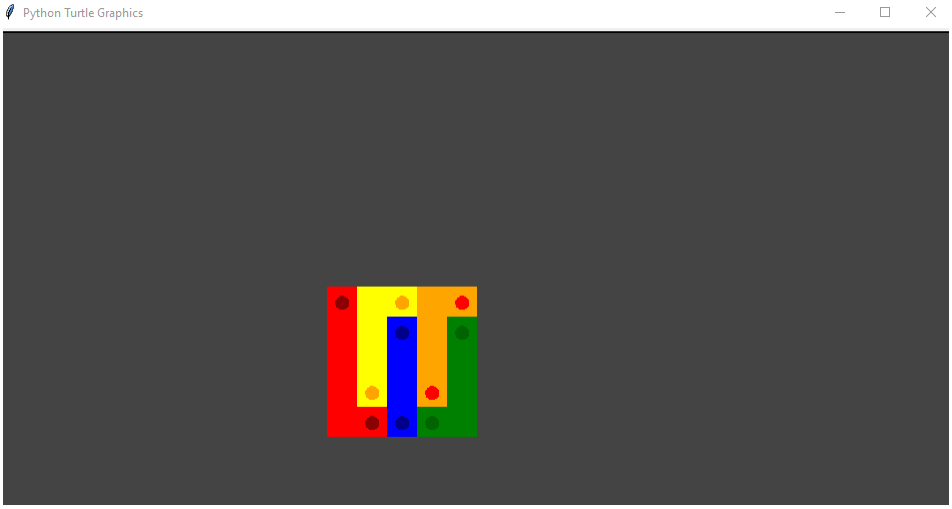




Resultado 3, Tablero 5x5




Python Turtle Graphics


Python Turtle Graphics