UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

DESARROLLO E IMPLEMENTACIÓN DE "PLUG-IN" PARA LA GENERACIÓN DE CARTODIAGRAMAS Y TIPOGRAMAS PARA QGIS3

TESIS
PARA OBTENER EL GRADO DE
INGENIERO EN REDES

PRESENTA
VÍCTOR JOSÉ LEAL PALACIOS

DIRECTORA
MSI. LAURA YÉSICA DÁVALOS CASTILLA

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
ÁREA DE TITULACIÓN

ASESORES
GEOG. ACATL GABRIEL REYES MEJÍA
MTI. MELISSA BLANQUETO ESTRADA
DR. JAVIER VÁZQUEZ CASTILLO
MTI. VLADIMIR VENIAMIN CABAÑAS VICTORIA

CHETUMAL QUINTANA ROO, MÉXICO, JUNIO DEL 2024

UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

TRABAJO DE TESIS TITULADO

"DESARROLLO E IMPLEMENTACIÓN DE "PLUG-IN" PARA LA GENERACIÓN DE CARTODIAGRAMAS Y TIPOGRAMAS EN QGIS3"

ELABORADO POR
VÍCTOR JOSÉ LEAL PALACIOS

BAJO SUPERVISIÓN DEL COMITÉ DEL PROGRAMA DE LICENCIATURA Y APROBADO COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:

INGENIERO EN REDES

COMITÉ DE TESIS

DIRECTORA: MS.I. LAURA YÉSICA DÁVALOS CASTILLA

ASESOR: GEOG. ACATL GABRIEL REYES MEJÍA

ASESORA: M.T.I. MELISSA BLANQUETO ESTRADA

ASESORA SUPLENTE: DR. JAVIER VÁZQUEZ CASTILLO

ASESOR SUPLENTE: MTI. VLADIMIR VENIAMIN CABAÑAS VICTORIA

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
ÁREA DE TITULACIÓN

CHETUMAL QUINTANA ROO, MÉXICO, JUNIO DEL 2024

Resumen

El primer capítulo introduce el problema que se busca resolver con la investigación, estableciendo la importancia y relevancia del tema en la ciencia de datos y la Geomática en el contexto actual. Se presenta la justificación y los objetivos generales y específicos del estudio.

El capítulo 2 presenta una revisión de literatura sobre la cartografía y los métodos de representación cartográfica, identificando sus limitaciones y las necesidades actuales. Se presenta un análisis detallado de los métodos de representación cartográfica convencionales y no convencionales, así como de las funciones de cartografía incorporadas en los Sistemas de Información Geográfica y los distintos tipos de software necesarios para la realización este trabajo de investigación. Se aborda también los métodos mediante los cuales se determinaron las clases y se calcularon los símbolos geométricos que conformaran los cartodiagramas y tipogramas.

En el capítulo 3 se describen los diferentes tipos de cartodiagramas y tipogramas que el plugin genera, se desarrolló del "Plug-in" describiendo los procesos y técnicas utilizadas para su creación, se presentan los casos de uso para el proceso de generación de los distintos cartodiagramas y tipogramas, así como las opciones de personalización de acuerdo con los datos que sé que necesiten representar visualmente.

En el capítulo 4 se discuten los resultados obtenidos con el desarrollo y la implementación del "Plug-in", evaluando su impacto en la representación y visualización de datos cartográficos y estadísticos.

AGRADECIMIENTOS

Quiero expresar mi sincero agradecimiento a todas aquellas personas que contribuyeron a la realización de este proyecto.

En primer lugar, agradezco a mis asesores, Lic. Acatl Gabriel Reyes Mejía, MTI Vladimir Vaniamin Cabañas Victoria, por su orientación y apoyo a lo largo de este proceso. Sus conocimientos y sugerencias fueron fundamentales para el desarrollo y éxito de este proyecto.

A mi directora, MSI Laura Yesica Dávalos Castilla, le agradezco profundamente por su guía constante, paciencia, motivación y confianza en mis capacidades.

Agradezco especialmente a mi esposa, Yarely Rivas Álvarez, por su amor, comprensión y apoyo incondicional durante estos años. Tu paciencia y aliento fueron la fuerza que necesitaba para alcanzar este logro.

También quiero expresar mi gratitud a mis padres, Víctor Hugo Leal Lucio y Xóchil Palacios Martínez, por su sacrificio y constante estímulo en mi desarrollo académico y personal.

Finalmente, agradezco a la MTI Melisa Blanqueto Estrada, al Dr. Javier Vázquez Castillo y a todos aquellos que de una u otra manera contribuyeron en este camino de aprendizaje y crecimiento profesional.

Este proyecto no habría sido posible sin el apoyo de cada uno. Gracias por ser parte de este importante logro académico.

Dedicatoria

A mi hija, aunque aún no has llegado a este mundo, ya llenas mi vida. Este proyecto no solo representa un logro académico, sino también un símbolo de amor y compromiso hacia ti. Eres la inspiración que impulsa mis sueños.

A mi amada esposa, Yarely Rivas Álvarez, quien ha sido mi apoyo incondicional en cada etapa de este viaje.

A mis padres, quienes me han enseñado el valor del esfuerzo y la dedicación. Su apoyo inquebrantable ha sido fundamental para alcanzar este logro.

A mis hermanos, por su aliento y comprensión, y por creer en mí incluso cuando las cosas parecían difíciles.

Contenido

CAPÍTULO 1

INTRODUCCIÓN

CAPÍTULO 1

1.1 Introducción

En el análisis espacial y la representación cartográfica, la visualización de datos geográficos es fundamental en la comprensión de patrones espaciales, la identificación de tendencias y la toma de decisiones. En ese sentido, los Cartodiagramas y Tipogramas son herramientas esenciales, permitiendo la representación visual de datos geográficos de manera efectiva y accesible para una amplia gama de usuarios.

Los Cartodiagramas, también conocidos como mapas temáticos o mapas de símbolos proporcionales, permiten representar la distribución espacial de un fenómeno mediante símbolos gráficos cuyo tamaño o color varía en función de los valores de los atributos asociados a las áreas geográficas correspondientes. Por otro lado, los Tipogramas, una forma especializada de Cartodiagrama, representan datos geográficos utilizando formas geométricas, como barras horizontales o verticales, cuyo tamaño se ajusta para reflejar la magnitud del fenómeno en estudio.

En este contexto, surge la necesidad de desarrollar herramientas que simplifiquen y mejoren el proceso de generación de cartodiagramas y tipogramas, permitiendo a los usuarios crear representaciones cartográficas y estadísticas de manera eficiente y personalizada. En respuesta a este problema, el presente trabajo se enfoca en el desarrollo e implementación de un plug-in para QGIS3, con el objetivo de proporcionar una solución accesible para la representación de datos geográficos.

El desarrollo de este plug-in no solo busca simplificar el proceso de creación de cartodiagramas y tipogramas, sino también ofrecer una amplia gama de opciones de personalización y configuración para adaptarse a las necesidades específicas de los usuarios y los requisitos de sus proyectos; brindando una herramienta versátil y eficaz para la exploración y análisis de datos geográficos en el software QGIS3x.

A través de la implementación de esta herramienta, se busca promover una mayor comprensión y comunicación de la información espacial, facilitando la toma de decisiones informadas y el análisis geoespacial en diversos campos de aplicación.

1.2 Justificación

La ciencia de datos es un campo compuesto de múltiples disciplinas que incluyen la Geomática, geografía, estadística, programación, análisis de datos, etc. y es una de las áreas que genera mayor interés y se prevé que sea de las más importantes en los próximos años.

Áreas como la Geomática, no disponen de recursos humanos capaces de desarrollar nuevas herramientas computacionales que les faciliten la representación y visualización de los datos geográficos y cartográficos que generan día a día. Al no existir personal que desarrolle estas herramientas, los geógrafos se limitan a obtener información de los datos visualizados o procesados, ya que es más difícil detectar patrones de comportamientos o comportamientos de los datos espaciales o territoriales. Estos además afectan en la falta de generación de políticas públicas encaminadas a solventar los patrones espaciales urbanos, por ejemplo, de las incidencias delictivas.

Adicionalmente, a esto: El volumen de información y la velocidad con la que esta se genera, nos ha obligado a desarrollar procesos para analizarla, procesarla y difundirla de forma más eficiente, se requiere el desarrollo de nuevas herramientas que permitan la visualización, el mejoramiento en la clasificación, orden y análisis de los datos para conocer mejor la realidad y focalizar los esfuerzos de la planeación, la inteligencia y la operatividad donde es necesario.

Los Sistemas de Información Geográfica (SIG), brindan una herramienta con la capacidad de analizar datos cartográficos-estadísticos complejos y generar conocimiento de manera óptima, sin embargo, sus procesos y métodos de

representación cartográfica, en su forma de construcción manual son realmente complejos y lentos.

Los métodos de representación cartográfica permiten expresar, transmitir, comunicar hechos y fenómenos geográficos del mundo real por medio de mapas. La naturaleza de estos hechos y fenómenos varía drásticamente, por lo cual se cuenta con una variedad de métodos de representación cartográfica capaces de representar cada fenómeno o hecho de la mejor forma posible y lograr la comprensión rápida y fácil del contenido del mapa. Considerando siempre los límites del método que se elija.

Los métodos de representación cartográfica convencionales son aquellos en los que solo se representa una variable de un hecho o fenómeno geográfico tales como método de puntos, líneas de flujo y cartogramas. Por lo que para una gran cantidad de fenómenos y hechos estos no son los adecuados para poder generar un mapa legible que permita analizar y comprender los patrones de distribución, variaciones y sus relaciones espaciales.

Los SIG tienen incorporados funciones de cartografía, pero adolecen o carecen de funciones no convencionales que son las que permiten visualizar múltiples datos, realizar representaciones de datos más complejas como las comparaciones numéricas multitemporales que proporcionan panoramas mucho más completos y pertinentes de los datos representados.

Gracias a las herramientas que la tecnología de SIG brinda, los métodos de representación cartográfica y procesos se han automatizado, lo cual permite generar modelos y mapas con una mayor rapidez y menor complejidad, pero a pesar de su creciente capacidad aún existen, procesos y métodos de representación cartográfica, que no están implementados, para su generación automática, además, debido a la complejidad que presentan estos para su construcción y análisis, los usuarios de Sistemas de Información Geográfica (SIG) no son recurrentes al uso de estos.

Por tales motivos, los aportes de este proyecto pretenden ayudar a desarrollar e implementar herramientas para la construcción de métodos de representación cartográfica, que no se encuentran disponibles en las herramientas de SIG, como se ha mencionado anteriormente.

El desarrollo e implementación se hará bajo una plataforma de software libre, lo cual permitirá a toda una comunidad de usuarios de disciplinas geográficas y ciencias de datos implementar proyectos innovadores para la visualización, manejo y representación cartográfica de datos con una fuerte componente territorial; proyectos que redundarán en un mejor acercamiento y conocimiento de la realidad geográfica, generando información concreta, oportuna para su modelación y la toma de decisiones.

1.3 Objetivo General

Desarrollar un "Plug-in" para el programa de sistemas de información geográfica de código abierto QGIS que permita la generación de métodos de representación cartográfica para representar gráficamente datos numéricos y estadísticos.

1.4 Objetivos particulares

- Analizar los métodos de representación cartográficos usados para representar cartográficamente datos numéricos y estadísticos
- Determinar los métodos de representación cartográfica que serán implementados en el "Plug-in".
- Identificar los procesos de construcción manual de los cartodiagramas que se implementaran en el "Plug-in".
- Definir una metodología de desarrollo de software.
- Desarrollo de un "plug-in" para la generación de métodos de representación cartográfica

1.5 Alcance

a. El proyecto solo abarca el procesamiento de capas vectoriales (puntuales, lineales y poligonales) libres de errores de topología.
b. Se generarán solo cartodiagramas de más de 3 variables.
c. No se contempla la creación de nuevas técnicas de representación cartográfica, solo se implementarán las existentes.

CAPÍTULO 2

MARCO TEÓRICO

CAPÍTULO 2

Es indispensable para el usuario y el desarrollador tener una base teórica clara sobre los cartodiagramas y mapas, para comprender los procesos y métodos que se utilizan en su construcción, así como las situaciones en las que podemos hacer uso de ellos.

Este capítulo tratara acerca de los cartodiagramas, dejando claro que existe una gran cantidad de estos y que además los usuarios pueden crear o usar diagramas personalizados, por lo cual, antes de proceder con el desarrollo del “Plug-in” debe analizarse qué tipos de cartodiagramas son los más apropiados antes de proceder a su construcción automatizada. Además, se definen los métodos matemáticos y geométricos que se usaron para la construcción de dichos cartodiagramas.

2.1 La Cartografía

Es el conjunto de estudios y operaciones científicas, artísticas y técnicas que intervienen, a partir de los resultados de las observaciones directas o de la explotación de una documentación, en el establecimiento de los mapas, planos y otras formas de expresión, así como su utilización. (Escobar, 2004).

2.2 Mapa

Existen varias definiciones acerca de lo que es un mapa geográfico. Se considera que la expresada por K. Salitchev es bastante precisa, ya que diferencia al mapa geográfico, del plano, del globo terráqueo, las imágenes de satélite o las fotografías, que son otras formas de representación de la tierra. K. Salitchev dice:

El mapa es una representación reducida, generalizada y matemáticamente determinada, de la superficie terrestre, sobre un plano, en el cual se interpreta la distribución, el estado y los vínculos de los distintos fenómenos naturales y socioeconómicos, seleccionados y caracterizados de acuerdo con la asignación concreta del mapa (Escobar, 2004)

A partir de esta definición, Salitchev señala las siguientes propiedades del mapa geográfico:

1. Las dimensiones de la tierra son de tal magnitud que no es posible representarla en su verdadero tamaño, sino que hay que reducirla miles o millones de veces, es decir, representarla a escala.
2. Respecto a la representación matemáticamente determinada de la superficie terrestre, sobre el plano, se refiere a la selección de la *proyección cartográfica* que sea más conveniente para elaborar un mapa, según si se requiere toda o parte de la superficie terrestre y del objetivo, contenido y usuario del mapa.
3. Se comprende como generalizada, el hecho de que no todos los elementos naturales, como ríos, relieve, costas y vegetación, entre otros; los culturales o humanos, como parcelas, puentes, presas, carreteras, edificaciones, asentamientos de población, etc., o los elementos abstractos, como número de habitantes, producción en toneladas, número de nacimientos o de migrantes, etc., se representan tal como son en la realidad, sino en forma esquemática, o por medio de signos o símbolos; es más, aquellos elementos naturales, culturales o abstractos poco significativos, no se llegan a representar en el mapa.

   La generalización tiene relación con la escala, el objetivo y la asignación del mapa.
4. Uso del lenguaje cartográfico, es decir, de signos, símbolos y color acordes con los fenómenos naturales, económicos, sociales, históricos o políticos que se requieran expresar en el mapa.
5. El método de representación cartográfica consiste en seleccionar el que sea conveniente para el lenguaje cartográfico o el contenido según las variables e indicadores.
6. La selección y generalización de los fenómenos que se representan.

Todas estas propiedades están estrechamente correlacionadas y son aplicables en la construcción y elaboración de los mapas.

2.3 Cartografía topográfica

La cartografía topográfica se ocupa de la elaboración de mapas en los que se representa en forma detallada y exacta la superficie terrestre, particularmente la altimetría y los accidentes de la superficie, así como la planimetría.

La elaboración de la carta topográfica se apoya en las fotografías aéreas, los levantamientos geodésicos, astronómicos, topográficos y de toponimias, así como en la restitución fotogramétrica. A la conjugación de estos apoyos se agregan a la proyección cartográfica y la escala del mapa. Esta carta permite obtener en forma precisa medidas de ángulos, distancias, áreas y desniveles.

La cartografía topográfica es de localización exacta, posición, altitud, forma, dimensiones e identificación de los accidentes del terreno, de los aspectos hidrológicos: ríos, lagos, lagunas; de la vegetación; de la planimetría o de los objetos materiales y culturales que se hallan de manera visible y permanente en el lugar: poblados, áreas construidas, infraestructura vial, presas puentes, escuelas, hospitales, líneas telefónicas, telegráficas, entre otros objetos. El contenido de esta carta se expresa por medio de símbolos y signos sencillos que están normalizados, es decir, tienen un uso y reconocimiento universal.

2.4 Cartografía temática

La cartografía temática trata sobre los aspectos teóricos, metodológicos y lógicos para la conceptualización y la elaboración de los mapas que no sean matemáticos ni topográficos, y su aplicación para guiar la elección y elaboración de los medios de expresión gráfica y las leyendas temáticas, en función de los fenómenos y de los objetos materiales o reales a representar.

En el mapa temático se representa en forma convencional, por medio de signos y símbolos cualitativos y/o cuantitativos, una información igualmente cualitativa y/o cuantitativa, correspondiente a los atributos o características concretas o abstractas de la componente “Z”, o sea de los hechos y fenómenos naturales, sociales, económicos, culturales, históricos o políticos que ocurren en el espacio geográfico.

Este mapa se elabora sobre un fondo geográfico de referencia, derivado de una carta topográfica de la cual se seleccionan y generalizan los elementos que se consideren indispensables, según el objetivo, la escala del mapa y la asignación del mapa temático, como son: las coordenadas, algunas curvas de nivel (generalmente las maestras), algunos ríos, lagos o presas; algunas vías de comunicación, algunas localidades con su ubicación puntual, o los limites político administrativos del nivel que se necesiten, entre otros elementos.

La información para la elaboración de un mapa temático se obtiene de diversas fuentes: material cartográfico, ya sea de contenidos físicos o humanos, observación directa de campo, información estadística y censal, encuestas, muestreo, fotografías aéreas e imágenes de satélite.

Cabe señalar que la expresión gráfica de la información debe ser legible, clara, precisa, a fin de que visualmente sea perceptible, comprensible y fácil y rápidamente se transmita al lector del mapa. Hay que agregar que los mapas son abstracciones y simplificaciones del espacio geográfico.

A diferencia de la cartografía topográfica, en la temática los elementos de expresión gráfica no son limitados, ni están normalizados, por el contrario, existe una gran diversidad de signos y símbolos, métodos de representación y formas de presentar los resultados hasta para un mismo tema, de acuerdo con la conceptualización y creatividad del autor del mapa.

Solamente se ha podido llegar a una normalización de los medios de expresión para algunos mapas temáticos del campo de la geografía física como los de vegetación, suelos, geología o hidrología.

Los temas a tratar en la cartografía temática son muy numerosos y variados, puesto que los fenómenos y hechos susceptibles de localizarse en los mapas son también innumerables, ya que este tipo de cartografía no se restringe solo a la elaboración de mapas referentes a fenómenos geográficos físicos o humanos, sino que puede hacerse extensiva a otros campos como el de la historia , la sociología, la demografía, la antropología, la política o hasta la estrategia militar, por citar algunos ejemplos.

Es conveniente destacar las posibilidades de elaborar un sinnúmero de mapas temáticos, ya que simplemente alrededor de un solo fenómeno geográfico de pueden tratar y expresar gráficamente diversos temas y hacer varias correlaciones, así como representar todos ellos de distintas formas.

2.5 El lenguaje Cartográfico

2.5.1 Componentes espaciales

Los hechos y fenómenos geográficos que interesan a los especialistas del espacio geográfico se pueden expresar y transmitir por medio de mapas.

El mapa se compone de una o varias imágenes. La imagen se crea y se lee en tres componentes espaciales; dos de los componentes de base del espacio geográfico, para responder a la pregunta geográfica ¿Dónde?, son las coordenadas geográficas o de localización: la longitud o componente “x” y la latitud o componente “y”.

Con estas coordenadas se identifican diversos puntos del plano en cuanto a su posición. El posicionamiento puede ser por punto, línea o área.

1. Puntual: Se define por un punto situado en el espacio geográfico.
2. Lineal: Resulta de la unión entre dos puntos del espacio geográfico.
3. Areal: Se define por la línea que une tres o más puntos.

La tercera componente, “Z”, se refiere a la cualidad o magnitud de los atributos de esos hechos y fenómenos, por ejemplo, la altitud, el relieve, el clima, el número de habitantes, la densidad de población, el área de un cultivo, etc.

2.5.2 Componentes del lenguaje cartográfico

Los componentes del lenguaje cartográfico Son: los símbolos (cualitativos y cuantitativos) combinados con las variables visuales y la forma de implantación grafica (puntual, lineal, areal). La combinación de estas tres componentes permite la expresión gráfica en el mapa de las componentes espaciales “x”, “y”, “Z”.

2.5.3 Símbolos y signos

Los objetos materiales o concretos como una ciudad, una escuela, hospital o fábrica, o los hechos abstractos como la distribución de la población analfabeta, rural, o el número de habitantes, es decir, los hechos y fenómenos geográficos no se representan tal y como son en la realidad, sino que se representan en forma esquematizada por medio de signos y símbolos figurativos y/o abstractos, cualitativos y/o cuantitativos como son figuras y dibujos de los objetos o símbolos abstractos como puntos, líneas, figuras geométricas y colores.

La necesidad de una representación simbólica es más evidente cuando se trata de variables o características abstractas, como una cantidad, un porcentaje, un carácter de la población, como la lengua o la religión.

Los símbolos en cartografía representan distribuciones, movimientos, desarrollo de un fenómeno, características cualitativas y cuantitativas y pueden ser simples o complejos.

Es conveniente utilizar símbolos claros, fáciles de leer, de dibujar y reducir a pequeñas dimensiones y que puedan ser reconocidos fácilmente sin rotulación.

2.5.4 Categorías de símbolos:

1. Símbolos convencionales fuera de la escala puntuales: Señalan el lugar en que se ubican los objetos que no pueden expresarse de acuerdo con la escala del mapa, ya que ocupan un área menor a la del signo.

2. Símbolos convencionales lineales: Señalan las líneas y objetos de extensión lineal.

3. Símbolos convencionales de superficie: Cubren un área y señalan características cualitativas y/o cuantitativas.

4. Símbolos evidentes o visuales: Son símbolos evocadores, esquematizados o simplificados, su localización y posición pueden ser sencillamente definida, y se asemejan a los objetos que representan.

5. Pictogramas: Son símbolos fáciles de comprender, evocan a un objeto o hecho de manera más real.

6. Ideogramas: Representan una idea.

7. Estarcido o trama: Es una estructura constituida por la repetición de un elemento grafico (líneas o puntos), de un símbolo o de un conjunto de símbolos sobre una superficie limitada.

8. Símbolos geométricos proporcionales: Estos símbolos son formas geométricas: círculos, esferas, cuadrados, cubos, rombos, rectángulos, triángulos, se diseñan fácilmente, ocupan poco espacio, el centro de la figura es en posición, por lo que indica exactamente la ubicación del hecho o fenómeno geográfico, además, estos símbolos pueden llevar color, textura e

incluso algunos se pueden orientar. Facilitan la lectura del mapa, hacer comparaciones e identificar variaciones y regularidades.

9. Símbolos de vectores, símbolos de movimiento o flechas: Estos símbolos son propios para representar fenómenos o hechos geográficos en movimiento.

10. Signos literales o alfanuméricos: Estos signos se emplean en mapas cualitativos.

2.5.5 La implantación grafica

La implantación grafica es la forma de representar los signos y símbolos combinados con las variables visuales sobre el mapa. La implantación depende del posicionamiento que tengan los hechos y fenómenos geográficos o los objetos materiales en el espacio geográfico o en el territorio la implantación de los símbolos puede ser puntual, lineal o areal.

2.5.6 Variables visuales

El lenguaje cartográfico es un lenguaje formal, sus procedimientos simbólicos de construcción, los elementos estructurales de las distribuciones, las relaciones entre los signos o símbolos y las distribuciones y la composición de las expresiones o imágenes se basan en principios básicos propuestos para la comunicación gráfica o semiología.

Al relacionar las características graficas de los signos o símbolos, que se reconocen e identifican, al expresarlos en los mapas con las características que se eligen de los datos o información de un lugar, se asigna un significado cualitativo y cuantitativo a esos símbolos, lo cual los convierte en símbolos designados. Asimismo, al disponer los símbolos en el plano horizontal X-Y se les da un significado geográfico, y lo expuesto en el plano se transforma en un mapa.

Para representar, expresar y comunicar los distintos datos de un modo concreto y claro mediante los símbolos y sus relaciones, se requiere modular, afinar, variar o suavizar la cualidad y percepción grafica de esos símbolos.

La modulación o variación se logra mediante la aplicación de las variables visuales o retinianas que se recomiendan y manejan en la comunicación gráfica: tamaño, intensidad, forma, orientación, textura o grano, color y tono o valor.

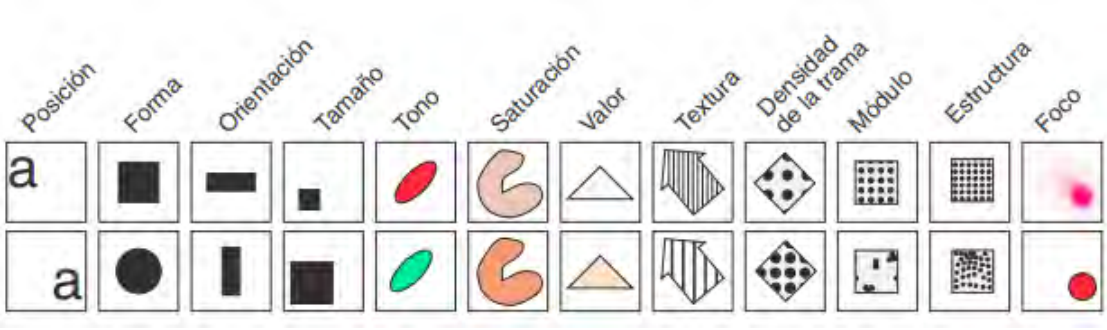

Figura 1 Variables visuales

2.6 Variables Visuales

2.6.1 Variable tamaño

Es una variable cuantitativa y expresa la proporción entre dos, tres o más magnitudes, esto es, el tamaño del símbolo varía proporcionalmente al valor de la magnitud o número absoluto o al valor relativo de los datos. Esta variable permite organizar los datos en una relación de ordenación progresiva ascendente o descendente y establece las diferencias y semejanzas de magnitud, por lo que también es una variable selectiva: "mayor que" o "menor que".

La función de la variable tamaño es crear la imagen, y por ello disocia cualquier otra variable con la que se combina.

2.6.2 Variable forma

La variable forma es ilimitada, por lo que existe una enorme posibilidad de crear formas, sin embargo, no se pueden dibujar numerosas formas bien diferenciadas, ni reproducirlas con rigor como el círculo, éste se percibe mejor que el cuadrado y éste mejor que el triángulo.

La variable forma permite la clasificación en grupos, de los hechos o fenómenos geográficos representados, así como hacer relaciones de semejanza-diferencia; es principalmente asociativa.

2.6.3 Variable orientación

La orientación es la variable visual que puede utilizarse con gran eficacia para representar algunos fenómenos dinámicos. Permite indicar la dirección hacia donde convergen o de donde divergen los fenómenos o hechos geográficos que se caracterizan por la movilidad espacial.

2.6.4 Variable textura o grano

La textura o grano se diseñan mediante elementos gráficos, ya sean líneas o puntos, que deben tener una variación sistemática tanto de grosor o tamaño como de espaciado, es decir, ambos aspectos deben tener regularidad.

La variable textura o grano se caracteriza porque la relación que establece es de semejanza- diferencia, por lo tanto, es selectiva y asociativa, esto es, por una parte, facilita separar áreas o zonas a través de varias intensidades selectivas y, por otra parte, agrupar en conjuntos semejantes.

La textura o grano también da lugar al nivel de organización ordenada, ya sea de manera cuantitativa o cualitativa, pues por sus características tiene la propiedad de clasificar y ordenar.

Esta variable se utiliza con frecuencia como símbolo de implantación areal o zonal y por su “transparencia” se puede representar en forma superpuesta, alguna

información adicional que guarde relación, procurando mantener la legibilidad del mapa.

2.6.5 Variable color

Esta variable visual es muy compleja, sin embargo, es muy usada por que es inmediata e intensamente perceptible, de tal forma que es bastante efectivo su uso después de la variable tamaño e intensidad ya que el color contribuye a la separación de imágenes y amplía las posibilidades de clasificación y de estructuración grafica jerarquizada, o sea la relación de orden y clasificación cuantitativa e incluso cualitativa, por lo que también permite establecer relaciones de semejanza y diferencia, de ahí que el valor de la percepción del color sea selectivo y asociativo.

El color relaciona conceptos y aspectos perceptivos psicológicos de importancia para la comunicación gráfica.

2.6.6 Variable valor

Esta variable se refiere al grado de percepción de oscuridad y claridad de un signo puntual, lineal o areal. Se aplica a las variables de color y de textura o grano, por ello se considera que su relación perceptiva es de semejanza-diferencia o sea selectiva, así como de orden o cuantitativa.

2.7 Métodos de representación cartográfica

2.7.1 Cartodiagramas y tipogramas

Con estos métodos se representa por medio de diagramas la distribución, más no la localización, de varios de los hechos o fenómenos geográficos cuantitativos discretos, ya sean sociales, económicos o físicos.

El Cartodiagrama es un método cartográfico de gran uso, su valor radica en que permite “combinar indicadores interrelacionados entre sí, en una misma figura,

mostrando las múltiples relaciones que se derivan del análisis conjunto del territorio y de la comparación de las unidades territoriales.

Los diagramas expresan el valor o la magnitud total, absoluta del hecho o fenómeno dentro de los límites territoriales de las unidades de representación.

Los símbolos o diagramas son de tamaño proporcional a los valores absolutos, dependiendo de la escala, del objetivo y asignación del mapa, así como la naturaleza de la información, puede ser en escala continua o en clases o intervalos regulares o irregulares, para este último tipo de clases el tamaño es proporcional a los limites superior de las clases o al punto medio de cada clase.

2.7.2 Tipos de diagramas

Lineales: Su longitud debe ser proporcional a los valores totales o absolutos que representan. Son útiles para comparar magnitudes en forma dinámica: gráficos de barras horizontales o verticales.

De superficie: también son proporcionales a los valores totales o absolutos que representan ocupan poco espacio y son fáciles de dibujar y manejar: círculos, rectángulos, triángulos, rombos.

De volumen: son proporcionales a los valores absolutos. Es conveniente usarlos cuando se tienen valores que oscilan fuertemente entre el máximo y mínimo.

2.8 Tipos de cartodiagramas y tipogramas

2.8.1 Diagramas simples.

Su tamaño es proporcional al volumen o valor absoluto de hecho o fenómeno geográfico que se representa. Al combinarse dos diagramas simples o más se puede mostrar la dinámica o evolución del fenómeno.

2.8.2 Diagramas simples con estructura

Su tamaño también es proporcional al volumen o valor absoluto de los hechos o fenómenos geográficos. Se dividen para expresar los valores relativos o la estructura de una serie de datos.

2.8.3 Diagramas que representan dos series de datos opuestos con o sin estructura

Cada serie de datos amplia y complementa la información. Los diagramas más usados son los círculos con o sin círculos internos y estos con o sin estructura, también los semicírculos unidos o separados por el diámetro permiten comparar valores de dos aspectos diferentes. También pueden construirse de manera que muestren la dinámica del fenómeno.

2.8.4 Diagramas complejos con varias series de datos

Los diagramas más utilizados son los cuadrantes de círculo o de cuadrados mediante los cuales se pueden representar cuatro series de datos opuestos que amplían y complementan la información sobre un fenómeno geográfico.

2.8.5 Diagramas complejos: combinados y con dinámica

Se pueden representar seis variables opuestas, complementarias y la dinámica de dos, tres, o más fenómenos para permitir observar su evolución; estos diagramas son llamados Tipogramas.

2.8.6 Otros diagramas son el de celdas, el triangular, y la pirámide de edades entre otros.

2.8.7 Ejemplos de cartodiagramas y tipogramas

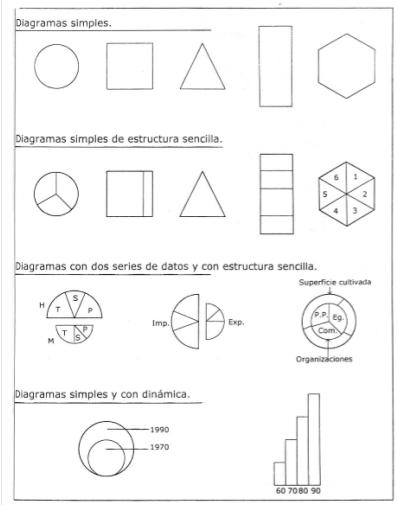

Figura 3 Diagramas simples

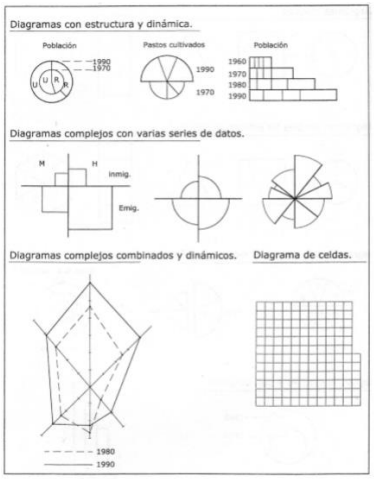

Figura 2 Diagramas con estructura compleja

2.9 Datos geográficos para elaborar mapas

Para expresar en un mapa los fenómenos y hechos geográficos y llegar a sus análisis, explicación y comprensión espacial se dispone de datos, observaciones o información de los mismos en cada punto, entre dos puntos o entre tres o más puntos del espacio geográfico; estos datos u observaciones se refieren a las características de cada punto del espacio: número de habitantes, población analfabeta, hombres, mujeres, ingresos, producción en toneladas, superficie cosechada, presión, lluvia, etcétera.

Clasificación de datos:

1. Cualitativa y cuantitativa.
2. Nominal, ordinal, por intervalos y de proporciones o razones.

Los datos cualitativos son aquellos que no son susceptibles de cuantificar mediante una escala numérica, sino que son categorías o modalidades

Los datos cuantitativos son los que pueden ser medidos o cuantificados numéricamente. Se clasifican en:

- Continuos, que toman un número infinito no numerable de valores posibles.

- Discretos, que solo pueden tomar un número finito, o infinito numerable de valores, con frecuencia solo absolutos o enteros.

Los datos cuantitativos individuales de las características de los fenómenos y hechos geográficos integran la serie de datos, dependiendo de la escala del mapa, del objetivo y asignación, estos datos pueden expresarse en él de dos formas:

1. Escala continua: Consiste en representar los valores de cada dato individual con símbolos cuyos tamaños varía de acuerdo con los valores de los datos de manera proporcional para darles tamaño conveniente, su uso es aceptable cuando se tienen muy pocos datos.

2. Clases: consiste en reducir el conjunto de los datos, agrupándolos, con lo cual el número de datos se simplifica y se pueden manejar y representar en el mapa más fácilmente. Se recomienda su uso cuando se tiene un gran número de datos.

Para agrupar los datos en “clases” se requiere del método que exprese mejor la diferencia en el territorio de los hechos y fenómenos geográficos.

Se recomienda que el número de clases sea impar, de preferencia de 3, o 5, ya que esto facilita la percepción visual selectiva en el mapa.

2.10 Métodos para determinar clases

Existen varios métodos para determinar las clases y sus límites: matemáticos y estadísticos.

Debe seleccionarse el método que mejor refleje la naturaleza de la distribución de los datos, de acuerdo con el propósito para el cual se requiere el establecimiento de las clases.

2.10.1 Progresión aritmética.

Se toma como razón aritmética para construir la progresión, el valor más bajo.

Ejemplo:
valores de la serie de datos 5, 7, 8, 10, 11, 12, 13, 14, 15, 16, 19, 20, 21, 22, 23, 25
El valor más bajo es 5

Tabla 1 Clases y frecuencia de datos

Progresión aritmética	Clases	Frecuencias de clases
5 + 5 = 10	5 – 10	4
10 + 5 = 15	11 – 15	5
15 + 5 = 20	16 – 20	3
20 + 5 = 25	21 – 25	4

Este método conviene cuando en la serie de datos los valores extremos (5 y 25) oscilan muy poco y el número de datos es reducido.

2.10.2 Progresión geométrica.

Este método se aplica cuando los valores de la serie de datos u observaciones, en general, son heterogéneos y los valores extremos oscilan fuertemente; Se toma como razón de la progresión geométrica el valor mínimo de la serie.

Ejemplo:
Valor mínimo de la serie: 10
Valor máximo de la serie: 1000000

Tabla 2 Clases y frecuencia de clases

Progresión geométrica	Clases	Frecuencias de clases
10 x 10 =100	10 – 100	8
100 x 10 = 1000	101 – 1000	10
1000 x 10 = 10000	1001 – 10000	15
10000 x 10 = 100000	10001 – 100000	12
100000 x 10 = 1000000	100001 – 1000000	9

2.10.3 Progresión geométrica II.

Otra forma para establecer clases por progresión geométrica, consiste en calcular la razón de la progresión mediante la fórmula:

$$a = \sqrt[n]{\frac{QM}{Qm}}$$

Donde QM = al valor máximo de la serie de datos

Qm = el valor mínimo de la serie de datos

n = el número de clases que se desean

a = la razón de la progresión

Ejemplo:

Valor mínimo de la serie: 9

Valor máximo de la serie: 6500

Numero de clases que se desean: 5

$$\sqrt[5]{\frac{6500}{5}} = 3.7$$

El valor de la razón es 3.7.

Tabla 3 Progresión geométrica II y clases

Progresión geométrica	Clases
9 x 3.7 = 33.3	9 – 33
33.3 x 3.7 = 123.2	34 – 123
123.2 x 3.7 = 455.8	124 – 456
455.8 x 3.7 = 1686.6	457 – 1687
1686.6 x 3.7 = 6240.9	1688 – 6500

2.10.4 Clases regulares u homogéneas o de intervalos iguales.

El intervalo se determina de la forma siguiente: se calcula la diferencia entre valor máximo y valor mínimo de la serie de datos, el resultado obtenido de la resta es el rango; este se divide entre el número deseado de clases.

Ejemplo:

Valor mínimo de la serie de datos: 8

Valor máximo de la serie de datos: 72

Numero clases deseado: 5

Intervalo de clase = $\frac{72-8}{5} = \frac{64(Rango)}{5} = 12.8$

Tabla 4 Clases regulares

Progresión	Clases
8 + 12.8 = 20.8	9 – 21
20.8 + 12.8 = 33.6	22 – 34
33.6 + 12.8 = 46.4	35 – 46
46.4 + 12.8 = 59.2	47 – 59
59.2 + 12.8 = 72	60 – 72

2.10.5 Clases irregulares con número igual de observaciones o datos cada clase.

Para obtener el número de clases se aplica la fórmula: 5 x log N. Donde N es el número de datos que integran la serie.

Ejemplo:
N: 60
Paso 1: obtener el número de clases aplicando la fórmula:
5 x log 60 = 8.89075
Este resultado indica que se pueden establecer entre ocho y nueve clases, con intervalos irregulares en cada una.

Pasó 2: se divide el número total de datos entre el número de clases:
Numero de clases: 8
60/8 = 7.5

Este resultado indica que cada clase debe tener siete u ocho datos, se toman 7 para el ejemplo. Los datos se colocan en orden descendente.

Tabla 5 Ordenamiento de datos y distribución equitativa por clases

Clases									
	1ra	2da	3ra	4ta	5ta	6ta	7ma	8va	9na
1	42	34	27	21	18	16	12	9	5
2	42	34	26	21	18	16	12	9	5
3	40	32	26	20	18	14	12	8	3
4	40	32	25	20	18	14	11	8	3
5	40	31	23	20	17	14	11	8	
6	39	30	23	20	17	14	11	7	
7	36	28	22	19	17	13	10	7	

Tabla 6 Clases irregulares

Clases definitivas
36 – 42
28 – 34
22 – 27
19 – 21
17 – 18
13 – 16
10 – 12
7 – 9
3 – 5

2.10.6 Para establecer límites de clases de intervalos irregulares o variables.

Los límites de clases varían en forma irregular, prácticamente en forma geométrica.

Ejemplo:

Se tienen 32 datos con valores que varían entre 130 (valor mínimo) y 363796 (valor máximo).

Paso 1: obtener el número de clases aplicando la fórmula:
5 x log 32 = 7.5

Este resultado indica que se pueden establecer entre siete y ocho clases, con intervalos irregulares en cada una.

Pasó 2: restar al logaritmo del valor máximo el logaritmo del valor mínimo:
Log 363796 = 5.5608
Log 130 = 2.1139
5.5608 – 2.1139 = 3.4469

Pasó 3: El resultado se divide entre el número de clases.
Numero de clases: 7
3.4469/7 = 0.4924 (este valor se considera como una constante).

Paso 4: Al logaritmo de valor máximo se le resta la constante, al resultado se le vuelve a restar la constante y así sucesivamente.

Tabla 7 Resta sucesiva de constante a logaritmo de valor máximo

Resta	Resultado
	5.5608(valor máximo)
5.5608 – 0.4924	5.0684

5.0684 – 0.4924	4.5760
4.5760 – 0.4924	4.0836
4.0836– 0.4924	3.5912
3.5912 – 0.4924	3.0988
3.0988 – 0.4924	2.6064
2.6064 – 0.4924	2.1147

Pasó 5: se busca el antilogaritmo de cada uno de los resultados y se usan como límites de las clases requeridas (el paso 4 puede hacerse también en forma inversa: sumando la constante al logaritmo del valor mínimo).

Tabla 8 Aplicación de antilogaritmo de resultados de la resta sucesiva de constante a logaritmo de valor máximo

Resultado	antilog
5.5608	363747.4
5.0684	117057.7
4.5760	37670.3
4.0836	12122.7
3.5912	3901.2
3.0988	1255.4
2.6064	404
2.1147	130

Tabla 9 Clases de intervalos variables

Clases definitivas
130 – 404
405 – 1255
1256 – 3901
3902 – 12123
12124 – 37670
37671 – 117057
117058 – 363796

2.11 Métodos para el cálculo de los símbolos geométricos

Los símbolos geométricos más comúnmente usados son los círculos, los cuadrados y los triángulos, ya que permiten variar su tamaño de acuerdo con la importancia cuantitativa del hecho o fenómeno o de sus características y con ello representar los valores de los datos de forma gráfica y visual de los mapas cuantitativos.

Los datos a representar pueden estar organizados en escala continua o en clases, según sea su naturaleza, su número, la escala del mapa y su objetivo y el usuario del mismo.

Los círculos, cuadrado y triángulos son símbolos cuya superficie debe ser proporcional a las entidades que representan.

El cálculo de su tamaño se basa en el método de la raíz cuadrada.

La superficie del circulo representa el valor absoluto, y para calcular su radio se aplica la fórmula $S = \pi R^2$, donde π es una constante que se puede suprimir.

El radio del círculo, por tanto, es proporcional a la superficie: $S = \sqrt{S}$. los resultados de esta fórmula pueden usarse como radios o como diámetros para la construcción de los círculos.

2.11.1 Círculos proporcionales

2.11.1.1 Método 1.

En principio se debe calcular la raíz cuadrada de todos los datos de la serie dada, y estos valores se dividen entre la raíz cuadrada del dato más pequeño.

Ejemplo:

Tabla 10 Datos y cálculo de radios de círculos.

Datos	$\sqrt{Dato}$	$R = \frac{\sqrt{dato}}{\sqrt{dato\ menor}}$
10126	100.6	1 mm
14124	118.8	1.2 mm
35840	189.3	1.9 mm
236520	486.3	4.8 mm
362720	602.2	5.9 mm
3580320	1892.2	18.8 mm
7420352	2724	27.0 mm

Si la longitud de los radios da lugar a círculos cuyo tamaño no es el adecuado para la escala del mapa, se puede aplicar entonces una constante (k) o coeficiente de proporcionalidad, la cual se escoge arbitrariamente, de manera que los valores de los radios permitan construir círculos de tamaños convenientes a la escala, manteniendo la proporcionalidad.

$$R = K\ \frac{\sqrt{dato}}{\sqrt{dato\ menor}}$$

2.11.1.2 Método 2.

Paso 1: Decidir el tamaño del circulo que se va a usar para representar el dato del valor más alto. Por ejemplo: un radio de 20 mm para el valor máximo de la serie de datos.

Paso 2: se calcula la raíz cuadrada de cada uno de los valores de la serie de datos.

Tabla 11 Datos y cálculo de constante y radios o diámetros de círculos.

Datos	$\sqrt{Dato}$	constante	Radio/diámetro
523422	723.5	0.3	21.7 mm
35280	187.8	0.3	5.6 mm
34174	184.9	0.3	5.5 mm
29279	171.1	0.3	5.1 mm
22866	151.2	0.3	4.5 mm

Paso 3: para calcular la longitud de cada unidad de radio o diámetro se divide el valor del circulo máximo asignado en el paso 1 entre la raíz cuadrada de cada dato o valor.

20 / 723.5 =0.027, aproximado 0.03; este valor se considera como una constante, por lo cual se multiplicarán los valores de las raíces cuadradas de cada dato de la serie.

2.11.1.3 Método 3. Círculos que representan datos en clases.

Paso 1: Establecer los intervalos

Tabla 12 Intervalos y cálculo de media para círculos

intervalos	Media
10 – 20	15
21 – 40	30
41 – 80	60
81 - 100	90

Paso 2: se obtiene la media o punto medio de cada intervalo.

Paso 3: a una unidad de superficie ($1\ mm^2\ o\ 1\ cm^2$) se le asigna un valor que se puede ir experimentando de acuerdo con el tamaño que se desea para los círculos. Por ejemplo $1\ cm^2$ = 5 toneladas. El valor 5 se toma como constante.

Paso 4: la media de cada clase se divide entre la constante, 5 en este caso.

Tabla 13 Cálculo de clases para círculos.

Media	constante	S=media/constante
15	5	3
30	5	6
60	5	12
90	5	18

Paso 5: se aplica a estos datos la siguiente fórmula.

Tabla 14 Clases y cálculo de radios de círculos.

clase	S	$R = \sqrt{\frac{S}{\pi}}$
Clase 1	3	0.97 cm

Clase 2	6	1.38 cm
Clase 3	12	1.95 cm
Clase 4	18	2.39 cm

2.11.1.4 Método 4. para obtener radio o diámetro de semicírculos.

Paso 1: Establecer los intervalos y se obtener la media o punto medio de cada intervalo

Tabla 15 Intervalos y cálculo de media para semicírculos

intervalos	Media
10 – 20	15
21 – 40	30
41 – 80	60
81 - 100	90

Paso 2: a una unidad de superficie ($1\ mm^2\ o\ 1\ cm^2$) se le asigna un valor que se puede ir experimentando de acuerdo con el tamaño que se desea para los círculos. Por ejemplo $1\ cm^2$ = 5 toneladas.

Pasó 3: el valor de la unidad de superficie se multiplica por dos, ya que el área del círculo está dividida a la mitad. $1\ cm^2$ = 5 x 2 = 10.
El valor 10 se toma como constante.

Paso 4: la media de cada clase se divide entre la constante, 10 en este caso.

Tabla 16 Cálculo de clases para semicírculos.

Media	constante	S=media/constante
15	10	1.5
30	10	3
60	10	6
90	10	9

Paso 5: se aplica a estos datos la siguiente fórmula.

Tabla 17 Clases y cálculo de radio para semicírculos.

clase	S	$R = \sqrt{\frac{S}{\pi}}$
Clase 1	1.5	0.7 cm
Clase 2	3	0.9 cm
Clase 3	6	1.4 cm
Clase 4	9	1.7 cm

2.11.1.5 Método 5. Para obtener radio o diámetro de los cuadrantes de círculos.

Paso 1: Establecer los intervalos y se obtener la media o punto medio de cada intervalo.

Tabla 18 Intervalos y cálculo de media para cuadrantes de círculo.

Intervalos	Media
10 – 20	15

21 – 40	30
41 – 80	60
81 - 100	90

Paso 2: a una unidad de superficie ($1\ mm^2\ o\ 1\ cm^2$) se le asigna un valor que se puede ir experimentando de acuerdo con el tamaño que se desea para los círculos. Por ejemplo $1\ cm^2$ = 5 toneladas.

Pasó 3: el valor de la unidad de superficie se multiplica por dos, ya que el área del círculo está dividida a la mitad. $1\ cm^2$ = 5 x 4 = 20.
El valor 20 se toma como constante.

Paso 4: la media de cada clase se divide entre la constante, 20 en este caso.

Tabla 19 Cálculo de clases para cuadrantes de círculo.

Media	constante	S=media/constante
15	20	0.7
30	20	1.5
60	20	3
90	20	4.5

Paso 5: se aplica a estos datos la siguiente fórmula.

Tabla 20 Clases y cálculo de radio de cuadrantes de círculo.

clase	S	$R = \sqrt{\frac{S}{\pi}}$
Clase 1	0.7	0.4 cm
Clase 2	1.5	0.7 cm

Clase 3	3	0.9 cm
Clase 4	4.5	1.2 cm

2.11.1.6 Método 6. Para obtener radio o diámetro de segmentos de círculos.

Paso 1: Establecer intervalos y se obtener la media o punto medio de cada intervalo.

Tabla 21 Intervalos y cálculo de media para segmentos de círculo.

Intervalos	Media
10 – 20	15
21 – 40	30
41 – 80	60
81 - 100	90

Paso 2: a una unidad de superficie ($1\ mm^2\ o\ 1\ cm^2$) se le asigna un valor que se puede ir experimentando de acuerdo con el tamaño que se desea para los círculos. Por ejemplo $1\ cm^2$ = 5 toneladas.

Pasó 3: el valor de la unidad de superficie se multiplica por la cantidad de segmentos deseado. por ejemplo 6 $1\ cm^2$ = 5 x 6 = 30.
El valor 20 se toma como constante.

Paso 4: la media de cada clase se divide entre la constante, 30 en este caso.

Tabla 22 Cálculo de clases para segmentos de círculo.

Media	constante	S=media/constante
15	30	0.5
30	30	1
60	30	2

90	30	3

Paso 5: se aplica a estos datos la siguiente fórmula.

Tabla 23 Clases y cálculo de radio de segmentos de círculo.

clase	S	$R = \sqrt{\frac{S}{\pi}}$
Clase 1	0.5	0.4 cm
Clase 2	1	0.5 cm
Clase 3	2	0.8 cm
Clase 4	3	1 cm

CAPÍTULO 3

DESARROLLO

CAPÍTULO 3

Este capítulo abordara definiciones, términos y conceptos relevantes, así como la metodología de prototipos evolutivos y su implementación en el desarrollo del “Plug-in”.

3.1 Ámbito de aplicación

3.1.1 Geomática

La Geomática es la disciplina que engloba las geociencias con la integración y aplicación de las tecnologías de la información y la comunicación (TIC). Esta suma de geociencias + TIC hace posible la captura, procesamiento, análisis, interpretación, almacenamiento, modelización, aplicación y difusión de la información digital geoespacial o localizada aplicable en los ámbitos de la ingeniería, el territorio y la sociedad. (http://geomaticaes.com/que-es-la-geomatica/)

3.1.2 Sistemas de información Geográfica (SIG)

Un SIG es un sistema que integra tecnología informática, personas e información geográfica, y cuya principal función es capturar, analizar, almacenar, editar y representar datos georreferenciados. (Olaya, 2014)

3.1.3 Modelo de datos

A pesar de la heterogeneidad de la información geográfica, existen dos aproximaciones básicas para simplificar y modelizar el espacio, de modo que este pueda ser almacenado y manipulado en un sistema informático, dando lugar, por tanto, a dos modelos de datos.

3.1.3.1 Modelo vectorial

El modelo vectorial se define por usar las figuras de la geometría convencional, puntos, líneas, curvas, polígonos, círculos, elipses o volúmenes para representar entidades del mundo real. (Jimenez, 2008)

3.1.3.2 Modelo raster

Se caracteriza por adoptar una unidad espacial estándar, el pixel, que no es sino un cuadrado, de tamaño elegible por el experto, que servirá para representar a un fragmento del espacio. (Jimenez, 2008)

3.2 QGIS

QGIS es un Sistema de Información Geográfica (SIG) de código abierto licenciado bajo GNU – General Public License. Es un proyecto oficial de Open Source Geospatial Fundation. Corre sobre Linux, Unix, Mac OSX, Windows y Android y soporta numerosos formatos y funcionalidades de datos vector, datos raster, y bases de datos (OSGEO, 2020).

Decenas de actualizaciones y mejoras cada cuatro meses, así como los cerca de 1000 complementos, contribuyen a su éxito; gracias al software libre cualquier persona que desarrolle un plug-in, puede subirlo al repositorio y ponerlo a disposición de todos los usuarios, además, los “plug-in” considerados de vital importancia pasan a ser parte del núcleo de QGIS.

3.2.1 PyQGIS

PyQGIS es una librería de QGIS para ejecutar código Python, con la cual, podemos automatizar tareas y desarrollar aplicaciones (complementos) mediante scripts. (Quiroz, 2020)

El desarrollo de complementos y aplicaciones personalizadas en PyQGIS está basado en el uso de la QGIS API (Interfaz de Programación de aplicaciones de QGIS, por sus siglas en ingles), la misma que cuenta con diferentes métodos que se integran en clases, las que a su vez están comprendidas en cinco módulos (Quiroz, 2020),

Módulos QGIS API:

- QGIS Core library: es la biblioteca central de QGIS y contiene toda la funcionalidad básica GIS.
- QGIS gui library: está construido en base a la biblioteca central y añade widgets de interfaz gráfica de usuario reutilizables.
- QGIS analysis library: construida en base a la biblioteca central, provee herramientas de alto nivel para llevar a cabo análisis espacial en datos raster y vectoriales.
- MapComposer: las clases del módulo de composición de mapas de QGIS incluyen métodos para el renderizado, creación de etiquetas, composición de leyenda, creación de grilla, visualización de archivos raster en el lienzo, visualización de caja de selección de colores y estilos entre otros.
- QGIS network analysis library: la biblioteca de análisis de redes provee herramientas de alto nivel para construir la topología y analizarla. (Quiroz, 2020)

3.2.2 Qt

Qt es un marco de desarrollo completo con herramientas diseñadas para agilizar la creación de aplicaciones e interfaces de usuario para plataformas de escritorio, integradas y móviles. (The Qt Company Ltd, 2020)

3.2.3 PyQt

PyQt es un conjunto de enlaces Python v2 y v3 para el marco de aplicaciones Qt y se ejecuta en toda la plataforma compatible con Qt. (Riverbank Computing, 2020)

3.2.4 Qt Designer

Qt Designer es la herramienta Qt para diseñar y construir interfaces graficas de usuario (GUI) con QtWidgets. Puede componer y personalizar sus ventanas o cuadros de dialogo. (The Qt Company Ltd, 2020).

3.3 Proceso de desarrollo de software

3.3.1 Programación orientada a objetos

La programación orientada a objetos (POO) es probablemente el paradigma más de moda hoy en día, y el que ha supuesto algunos de los mayores avances en los últimos tiempos.

En este paradigma se trabaja con objetos, que son estructuras que permiten tanto almacenar como manipular datos de forma coherente e identificada.

3.3.2 Python

Python es un lenguaje interpretado, de alto nivel y enfocado principalmente a la legibilidad y facilidad de aprendizaje y uso. (Gutierrez, 2016)

Es un lenguaje orientado a objetos, aunque soporta otros paradigmas como la programación funcional y, por supuesto, la programación imperativa. (Gutierrez, 2016)

Características:

- Simplicidad
- Sintaxis clara
- Propósito general
- Software Libre
- Extensas librerías
- Incrustable

3.3.3 Metodología de Prototipos evolutivos.

Se basa en la creación de una implementación parcial de un sistema, para el propósito explícito de aprender sobre los requerimientos del sistema, un prototipo es construido de una manera rápida tal como sea posible. Esto es dado a los

usuarios, clientes o representantes de ellos, posibilitando que ellos experimenten con el prototipo. Estos individuos luego proveen la retroalimentación sobre lo que ellos les gusto y no les gusto acerca del prototipo proporcionado, quienes capturan en la documentación actual de la especificación de requerimientos la información entregada por los usuarios para el desarrollo del sistema real.

Fases del método de prototipos evolutivos:

Fase de investigación preliminar.

Las metas principales de esta fase son: Determinar el problema y su ámbito, la importancia y efectos sobre la organización por una parte y, por otro lado, identificar una idea general de la solución para realizar un estudio de factibilidad que determine la viabilidad de una solución software.

Esta primera fase se divide en tres actividades:

- Clasificación de requerimientos
- Estudio de factibilidad
- Aprobación del requerimiento

Fase de definición de requerimientos del sistema.

El objetivo de esta etapa es registrar todos los requerimientos y deseos que los usuarios tienen en relación con el proyecto bajo desarrollo. Esta etapa es la más importante del modelo, es aquí donde el desarrollador determina los requisitos mediante la construcción, demostración y retroalimentaciones del prototipo.

Consiste en cinco etapas:

Análisis grueso y especificación
Diseño y construcción
Evaluación
Modificación

Termino

Fase de diseño Técnico.

Durante la construcción del prototipo, el desarrollador no ha realizado el diseño detallado. El sistema entonces debe ser rediseñado y documentado según los estándares de la organización y para ayudar al mantenimiento del sistema:

Consiste en dos etapas:

Producción de documentación de diseño

Producción de requerimientos para realizar mantenimiento futuro al software.

Fase de desarrollo y pruebas.

Los cambios son identificados en el diseño técnico, son implementados y probados para asegurar la corrección y completitud de estos con respecto a los requerimientos.

Fase de operación y mantenimiento.

Consiste en la instalación del sistema en ambiente del usuario, en este caso, resulta de menor complejidad, ya que se supone que los usuarios han trabajado con el sistema al hacer pruebas de prototipos. Considera al mantenimiento una fase de menor importancia, ya que a lo largo del desarrollo del proyecto de corrigieron los errores del prototipo, por lo cual el mantenimiento correctivo del software se reduce. Si eventualmente se requiere dar mantenimiento entonces el proceso de prototipado es repetido definiendo un nuevo conjunto de requerimientos (G, 2020).

3.4 Fase de investigación preliminar

En esta fase determinamos el problema, para obtener una idea general de la solución y cuál es el grado de viabilidad para desarrollar el “Plug-in”, asimismo identificamos los requerimientos iniciales del usuario, es decir las tareas que realizara el “Plug-in”.

3.4.1 Clarificación de requerimientos.

En esta etapa definimos cuales son los requerimientos iniciales del usuario, tomando como base los objetivos planteados para este proyecto.

Para llevar a cabo esta etapa se clasificaron los requerimientos en dos grupos:

- Requerimientos funcionales: son las tareas que el “Plug-in” realiza.
- Requerimientos no funcionales: son atributos que presenta el “Plug-in” pero que no son una funcionalidad especifica.

3.4.1.1 Requerimientos Funcionales.

El “Plug-in” debe permitir al usuario procesar datos de capas vectoriales: puntuales, lineales y poligonales.

El “Plug-in” debe construir cartodiagramas y tipogramas complejos con varias series de datos, combinados y con dinámica.

El “Plug-in” deberá proveer una interfaz gráfica donde el usuario pueda seleccionar:

1. Tipo de Cartodiagrama o Tipograma.
2. Datos con los que se construyen los Cartodiagramas y Tipogramas
3. Color y ancho de línea de los Cartodiagramas y Tipogramas.
4. Color de relleno de los Cartodiagramas y Tipogramas.
5. Opacidad del color de relleno.

El “Plug-in” deberá generar un Cartodiagrama y un Tipograma.

1. Cartodiagrama estructurado
2. Tipograma lineal dinámico

La opción Cartodiagrama estructurado deberá generar cartodiagramas de círculos simples, ángulos fijos, ángulos variables, semicírculos, sectores y la combinación de estos.

La opción Tipograma lineal dinámico deberá generar un Tipograma de ejes fijos.

Las distintas variaciones de los cartodiagramas y tipogramas se generarán de acuerdo a los parámetros disponibles que el usuario podrá modificar en el “plug-in”.

3.4.1.2 Requerimientos no funcionales.

- El “Plug-in” debe ser lo más intuitivo posible y fácil de usar.
- El “Plug-in” debe poder integrarse a QGIS 3.10.
- El “Plug-in” debe ser desarrollado en lenguaje Python.

3.4.2 Estudio de factibilidad.

En esta etapa se analiza el grado de factibilidad para llevar a cabo el proyecto, considerando tres aspectos básicos:

3.4.2.1 Factibilidad técnica.

Capacidad de personal.

Tabla 24 Componentes y nivel de conocimientos necesario del personal.

Componentes	Nivel de Conocimiento
Conocimiento de cartografía	Básico
Manejo de QGIS 3.10	Básico

Hardware.

Tabla 25 Requerimientos de hardware.

PC/Laptop	
Componentes	Requerimientos mínimos
Procesador	Intel Core I3
Memoria RAM	4 GB
Capacidad Almacenamiento	160 GB
Teclado	Genérico
Mouse	Genérico
Monitor (PC)	Genérico

Software.

Tabla 26 Requerimientos de software

Componentes	Requerimientos mínimos
Sistema Operativo	Windows 10 64 bits
SIG	QGIS 3.10

Las capacidades de personal actuales de la institución son las adecuadas para poder llevar a cabo el proyecto, los usuarios del “plug-in” tienen los conocimientos necesarios tanto en cartografía como en el manejo de QGIS 3.10, el cual ya se encuentra instalado y ejecutándose correctamente en los equipos actuales.

Por lo tanto, basándonos en la factibilidad técnica, podemos decir que es factible realizar el proyecto.

3.4.3 Factibilidad económica.

Costos de Personal. No se estiman costos de personal, debido a que se cuenta con el personal suficiente y con los conocimientos necesarios para llevar a cabo el proyecto.

Costos de Hardware. Se cuenta actualmente con el equipo necesario y con los requerimientos mínimos, por lo que los costos asociados a este aspecto son nulos.

Costos de software. Los equipos ya cuentan con el software apropiado instalado, además de que QGIS 3.10 es software libre, lo cual no supone ningún costo o gasto extra, por lo que podemos decir que los costos asociados a software son nulos.

En cuanto a la factibilidad económica, el proyecto es realmente viable debido que los costos son prácticamente nulos, aunado a que el con el desarrollo de este proyecto se obtendrá un beneficio alto al ahorrar tiempo y recursos.

3.4.4 Factibilidad operativa.

Debido a que no existe una herramienta similar que se esté utilizando, no se espera resistencia al cambio por parte de los usuarios.

3.5 Fase de definición de requerimientos del sistema.

El objetivo de esta etapa es registrar todos los requerimientos y deseos que los usuarios tienen con relación al proyecto bajo desarrollo. Esta etapa es la más importante del modelo, es aquí donde el desarrollador determina los requisitos mediante la construcción, demostración y retroalimentaciones del prototipo.

3.5.1 Análisis grueso y especificación.

3.5.1.1 Diagramas de casos de uso.

Los requerimientos de las funciones que realiza el “Plug-in” se representaran detalladamente mediante casos de uso. La Figura 4 muestra los casos de uso del contexto general del “Plug-in”.

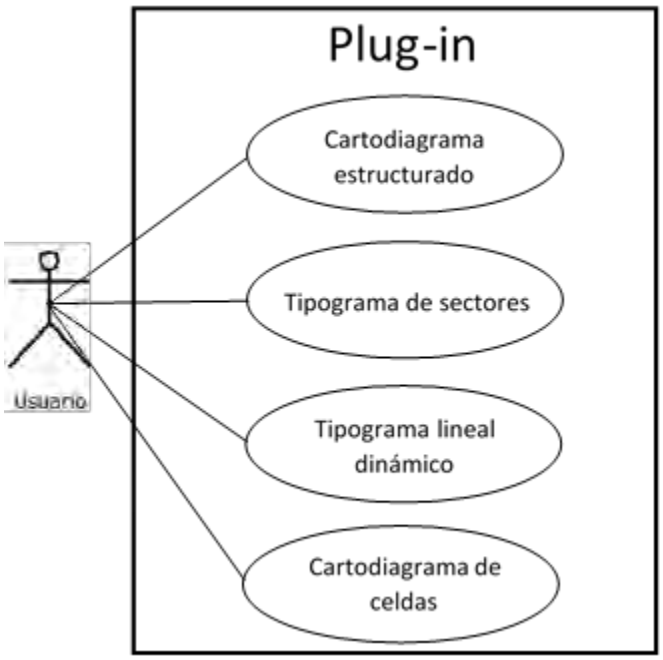

Figura 4 Diagrama de casos de uso

Cartodiagrama estructurado.

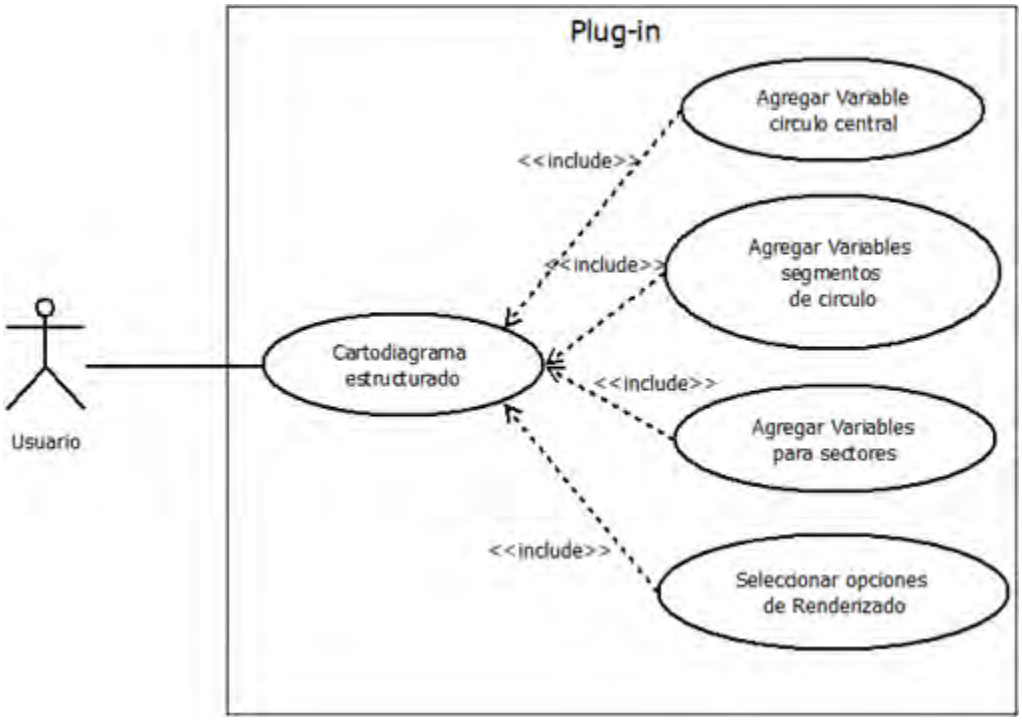

Figura 5 Diagrama de caso de uso Cartodiagrama estructurado

Tabla 27 Detalles de caso de uso Cartodiagrama estructurado

Detalles de caso de uso

Nombre caso de uso	Cartodiagrama estructurado
Actores	Usuario y "Plug-in"

Objetivo

Construir Cartodiagrama estructurado

Descripción

Construye el Cartodiagrama estructurado, de acuerdo con las opciones elegibles por el usuario que ofrece el "Plug-in".

Precondiciones

- *El usuario debe agregar las variables a aplicar.*
- *El usuario debe agregar 2 variables para los segmentos de círculo.*
- *Deben existir las clases o estar seleccionadas la opción de escala continua.*

Post-condiciones

- *Se construye el Cartodiagrama estructurado.*
- *La cantidad de sectores en cada segmento de círculo estará determinada por el número de variables de sectores agregadas.*

Flujo del "Plug-in"

Paso	**Acción**
1	El usuario seleccionar la opción Cartodiagrama estructurado.
2	El usuario agrega la variable a aplicar para el círculo central. Ver caso de uso agregar variable círculo central
3	El usuario agrega las variables para los segmentos de círculo. Ver caso de uso agregar variables segmentos de círculo.
4	El usuario Agrega Variables para sectores. Ver caso de uso agregar variable para sectores.
5	El usuario Selecciona opciones de renderizado. Ver caso de uso seleccionar opciones de renderizado
6	El usuario aplica los cambios.

7	El “Plug-in” calcula el tamaño de los segmentos de circulo: 7.1 Si el usuario selecciono escala continua se omite el uso de clases y se calcula el tamaño mediante el método 2 de círculos proporcionales. El valor para tamaño máximo es seleccionado por el usuario. Ver caso de uso seleccionar opciones de renderizado. 7.2 se calcular el tamaño mediante Método 4 para obtener radio o diámetro de semicírculos.
8	El “Plug-in” construye los segmentos de círculo: 8.1 Si el usuario agrego variables para sectores, se calcula el ángulo de los sectores mediante el método de cálculo de ángulo de sectores. 8.2 El “plug-in” construye los sectores. 8.3 Si el usuario no agrego variables para sectores se construyen los segmentos de círculo de acuerdo al tamaño calculado en el paso 7.
9	El “Plug-in” calcula el tamaño del círculo central: 9.1 Si el usuario selecciono escala continua se omite el uso de clases y se calcula el tamaño mediante el método 2 de círculos proporcionales. El valor para tamaño máximo es seleccionado por el usuario. Ver caso de uso seleccionar opciones de renderizado. 9.2 Se calcular el tamaño mediante el Método 3 Círculos que representan datos en clases. 9.3 El “plug-in” construye el círculo central.
10	Fin de caso de uso

Observaciones

- *El usuario puede omitir el paso 5 debido a que el “Plug-in” ya tiene valores por defecto para estas opciones.*

Tipograma de sectores.

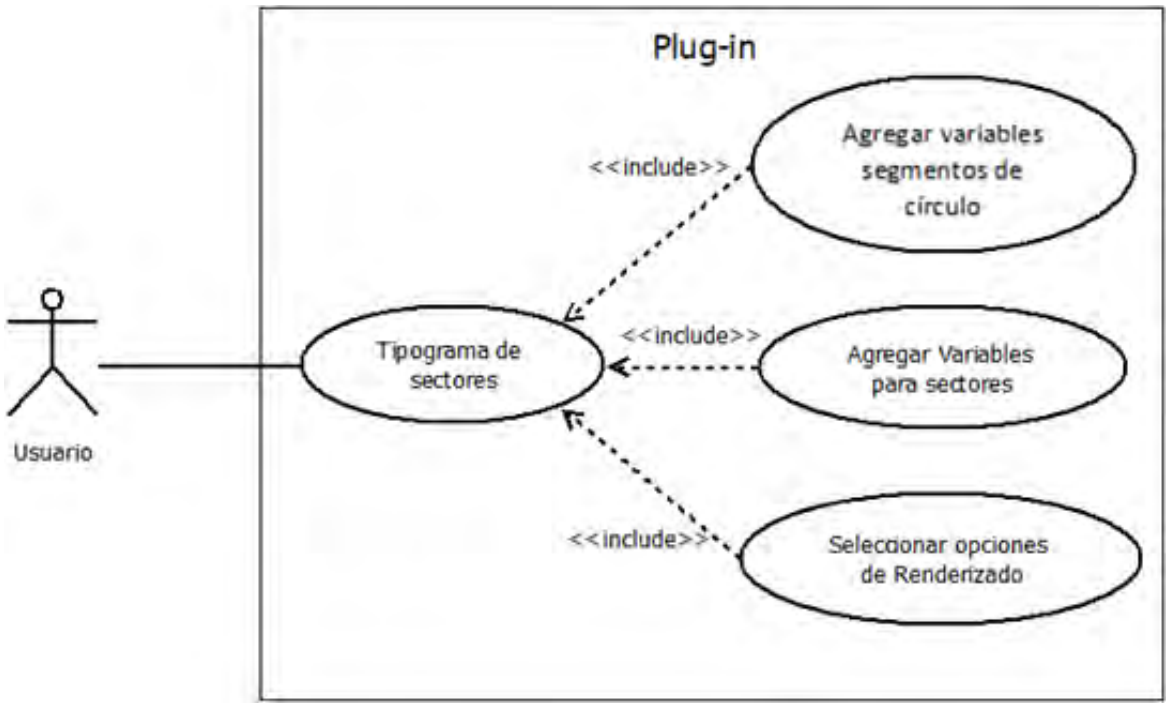

Figura 6 Diagrama de caso de uso Tipograma de sectores

Tabla 28 Detalles de caso de uso Tipograma de sectores

Detalles de caso de uso

Nombre caso de uso	Tipograma de sectores
Actores	Usuario y "Plug-in"

Objetivo

Construir Tipograma de sectores

Descripción

Construye el Tipograma de sectores, de acuerdo a las opciones elegibles por el usuario que ofrece el "Plug-in".

Precondiciones

- *El usuario debe agregar las variables a aplicar.*
- *El usuario debe agregar 2 o más variables para los segmentos de círculo.*
- *Deben existir las clases o estar seleccionada la opción de escala continua.*

Post-condiciones

- *Se construye el Tipograma de sectores*
- *La cantidad de sectores en cada segmento de círculo estará determinada por el número de variables de sectores agregadas.*

Flujo del "Plug-in"

Paso	**Acción**
1	El usuario seleccionar la opción Tipograma de sectores.
2	El usuario agrega las variables para los segmentos de círculo. Ver caso de uso agregar variables segmentos de círculo.
3	El usuario Agrega Variables para sectores. Ver caso de uso agregar variable para sectores.
4	El usuario Selecciona opciones de renderizado. Ver caso de uso seleccionar opciones de renderizado
5	El usuario aplica los cambios.
6	El "Plug-in" calcula el tamaño de los segmentos de circulo:

	4.1 Si el usuario selecciono escala continua se omite el uso de clases y se calcula el tamaño mediante el método 2 de círculos proporcionales. El valor para tamaño máximo es seleccionado por el usuario. Ver caso de uso seleccionar opciones de renderizado. 4.2 se calcular el tamaño mediante Método 5 Para obtener radio o diámetro de segmentos de círculos.
7	El "Plug-in" construye los segmentos de círculo: 7.1 Si el usuario agrego variables para sectores, se calcula el ángulo de los sectores mediante el método de cálculo de ángulo de sectores. 7.2 El "plug-in" construye los sectores. 7.3 Si el usuario no agrego variables para sectores se construyen los segmentos de círculo de acuerdo al tamaño calculado en el paso 6.
8	Fin de caso de uso

Observaciones

- *El usuario puede omitir el paso 4 debido a que el "Plug-in" ya tiene valores por defecto para estas opciones.*

Agregar variable círculo central.

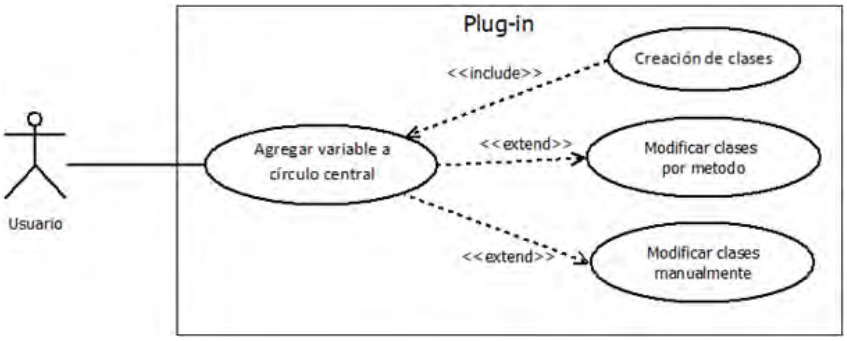

Figura 7 Diagrama de caso de uso agregar variable a círculo central.

Tabla 29 Detalles de caso de uso agregar variable de circulo central.

Detalles de caso de uso

Nombre caso de uso	Agregar variable a círculo central
Actores	Usuario y "Plug-in"

Objetivo

Agregar variable para construir el círculo central del Cartodiagrama estructurado

Descripción

Se selecciona y se agregar la variable que se usara para construir el círculo central del Cartodiagrama estructurado. Así también el "Plug-in" creara las clases para calcular el tamaño del círculo central del Cartodiagrama estructurado.

Precondiciones

- *El usuario debe seleccionar una variable.*

Post-condiciones

- *Se agrega la variable.*
- *Se crean las clases.*

Flujo del "Plug-in"

Paso	**Acción**
1	El usuario agrega la variable.
2	El "Plug-in" genera las clases. Ver caso de uso creación de clases
3	El usuario tiene la opción de modificar las clases por medio de un método seleccionable. Ver caso de uso modificar clases por método.
4	El usuario tiene la opción de modificar las clases manualmente. Ver caso de uso modificar clases manualmente
5	Fin de caso de uso

Observaciones

Agregar variables segmentos de círculo.

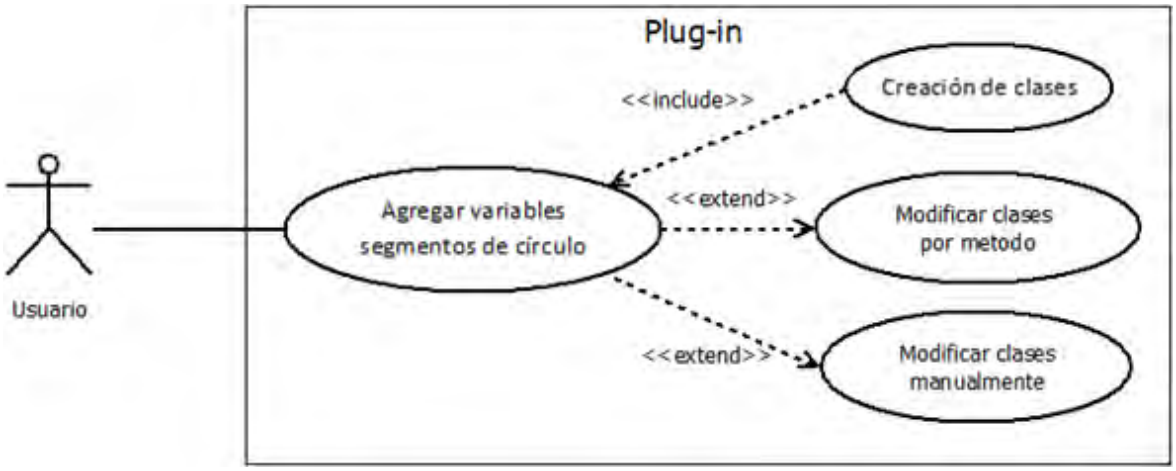

Figura 8 Diagrama de caso de uso agregar variables para segmentos de círculos.

Tabla 30 Detalles de caso de uso agregar variables para segmentos de circulo

Detalles de caso de uso

Nombre caso de uso	Agregar variables para segmentos de circulo
Actores	Usuario y "Plug-in"

Objetivo

Agregar variables para construir los segmentos de círculo.

Descripción

Se seleccionan y se agregan las variables que se usaran para construir los segmentos de círculo. Así también el "Plug-in" creara las clases para calcular el tamaño de los segmentos de circulo

Precondiciones

- *El usuario debe seleccionar una variable.*

Post-condiciones

- *Se agregan las variables.*
- *Se crean las clases.*

Flujo del "Plug-in"

Paso	**Acción**
1	El usuario agrega las variables.
2	El "Plug-in" genera las clases. Ver caso de uso creación de clases
3	El usuario tiene la opción de modificar las clases por medio de un método seleccionable. Ver caso de uso modificar clases por método.
4	El usuario tiene la opción de modificar las clases manualmente. Ver caso de uso modificar clases manualmente
5	Fin de caso de uso

Observaciones

Agregar variables para sectores

Detalles de caso de uso

Nombre caso de uso	Agregar variables para sectores
Actores	Usuario y “Plug-in”

Objetivo

Agregar variables para construir los sectores de los segmentos de círculo.

Descripción

Se seleccionan y se agregan las variables que se usaran para construir los sectores de los segmentos de círculo. Así también el “Plug-in” almacena estas variables para poder modificarlas posteriormente.

Precondiciones

- *El usuario debe seleccionar una variable.*

Post-condiciones

- *Se agregan las variables.*
- *Se almacena las variables.*

Flujo del “Plug-in”

Paso	**Acción**
1	El usuario agrega las variables.
2	El “Plug-in” almacena las variables.
3	Fin de caso de uso

Observaciones

Seleccionar opciones de renderizado

Detalles de caso de uso

Nombre caso de uso	Seleccionar opciones de renderizado
Actores	Usuario y "Plug-in"

Objetivo

Editar las opciones de renderizado que ofrece el "Plug-in"

Descripción

Se modifican los valores de las opciones de renderizado que se usaran para construir los Cartodiagrama y Tipograma.

Precondiciones

- *El usuario debe seleccionar la opción de renderizado.*

Post-condiciones

- *Los Cartodiagrama y Tipograma serán construidos con los valores de estas opciones.*

Flujo del "Plug-in"

Paso	**Acción**
1	El usuario modifica los valores de las siguientes opciones que ofrece el "Plug-in": • Transparencia: Modifica la transparencia del color de relleno de los cartodiagramas y Tipograma. • Color de línea: Modifica el color con el que se dibujaran las líneas de los cartodiagramas y Tipograma. • Ancho de línea: Modifica el ancho de línea con el que se dibujaran las líneas de los cartodiagramas y Tipograma. • Ajuste de tamaño de círculo central: Modifica el valor que se usa para calcular el tamaño del círculo central en el caso de uso Cartodiagrama estructurado.

	• Ajuste de tamaño de segmentos de círculo: Modifica el valor que se usa para calcular el tamaño de los segmentos de círculo. • Distancia: Modifica el valor de la distancia entre segmentos de circulo.
2	Fin de caso de uso

Observaciones

Creación de clases

Detalles de caso de uso

Nombre caso de uso	Creación de clases
Actores	“Plug-in”
Objetivo	
Crear las clases que se usaran para construir los Cartodiagrama y Tipograma	
Descripción	
Se crean las clases de acuerdo a las variables que el usuario agrego	
Precondiciones	
• **El usuario debe agregar al menos una variable.**	
Post-condiciones	
• **Se generarán las clases.** • **El usuario podrá editar las clases.**	
Flujo del “Plug-in”	
Paso	**Acción**
1	El “Plug-in” carga los datos de la variable agregada
2	El “Plug-in” genera las clases mediante el método para establecer límites de clases de intervalos irregulares o variables.
3	El “Plug-in” almacena las clases generadas.
4	Fin de caso de uso
Observaciones	

Modificar clases por método

Detalles de caso de uso

Nombre caso de uso	Modificar clases por método
Actores	Usuario y "Plug-in"

Objetivo

Modificar las clases que se usaran para construir los Cartodiagrama y Tipograma

Descripción

Se modifican las clases de acuerdo a las variables que el usuario agrego y al método de construcción de clases que el usuario seleccione.

Precondiciones

- *El usuario debe agregar al menos una variable.*

Post-condiciones

- *Se modificaran las clases.*

Flujo del "Plug-in"

Paso	**Acción**
1	El usuario selecciona la opción de clases
2	El usuario selecciona uno de los métodos de construcción de clases: a) Progresión aritmética b) Progresión geométrica c) Progresión geométrica II d) Clases regulares u homogéneas o de intervalos iguales e) Clases irregulares con número igual de observaciones o datos cada clase f) Para establecer límites de clases de intervalos irregulares o variables
3	El "Plug-in" genera las clases mediante el método elegido.
4	El "Plug-in" almacena las clases generadas.

5	Fin de caso de uso

Observaciones

Modificar de clases manualmente

Detalles de caso de uso

Nombre caso de uso	Modificar clases manualmente
Actores	Usuario y “Plug-in”
Objetivo	
Modificar las clases que se usaran para construir los Cartodiagrama y Tipograma	
Descripción	
El usuario modifica manualmente las clases.	
Precondiciones	
• **El usuario debe agregar al menos una variable.**	
Post-condiciones	
• **Se modificarán las clases.**	
Flujo del “Plug-in”	
Paso	**Acción**
1	El usuario selecciona la opción de clases.
2	El usuario selecciona un intervalo.
3	El “Plug-in” solicita al usuario un valor inicial y un valor final para el intervalo.
3	El “Plug-in” almacena el intervalo.
4	Fin de caso de uso
Observaciones	
• **El usuario puede eliminar los intervalos existentes y agregar nuevos.** • **Al agregar un nuevo intervalo el “Plug-in” solicitara al usuario un valor inicial y un valor final.**	

Diagrama de clases

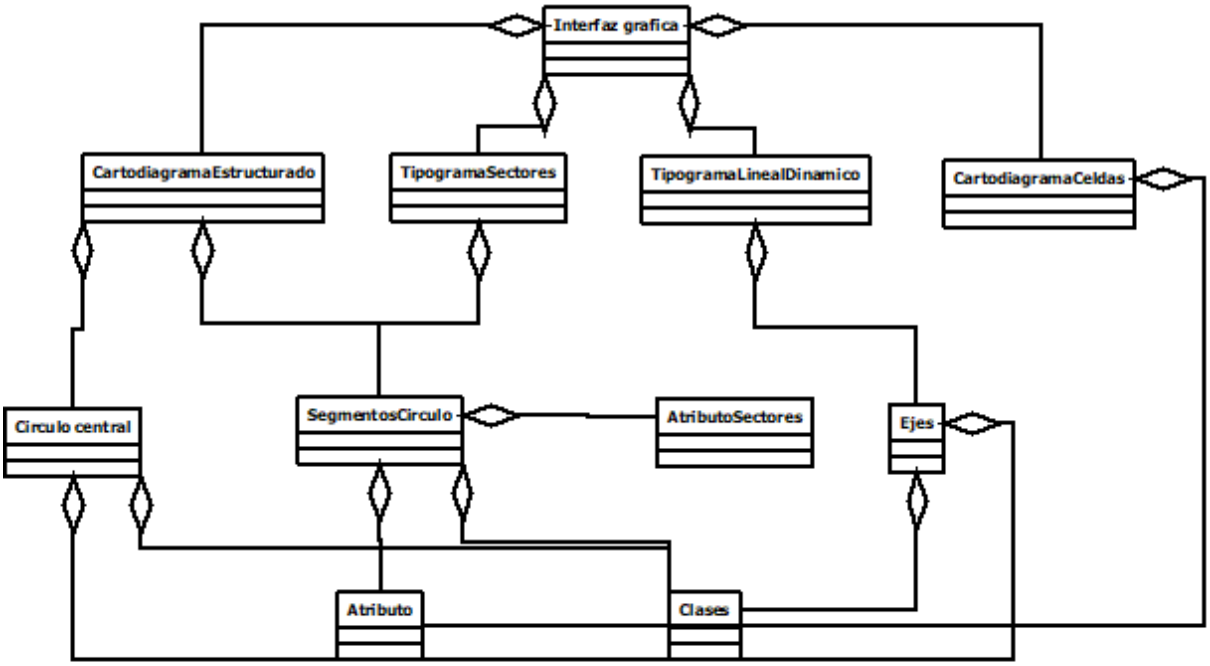

Figura 9 Diagrama de clases para el desarrollo del plug-in.

3.4.2 Diseño y construcción .

Esta etapa consiste en obtener un prototipo inicial con máxima funcionalidad. Poniendo énfasis en la interfaz de usuario.

3.4.2.1 Interfaz de usuario .

La interfaz de usuario se compone de una ventana Inicial Figura 10Figura 10 Ventana principal, en la cual se pueden visualizar cinco pestañas, el selector del tipo de Cartodiagrama o Tipograma y los botones para aplicar o cancelar los cambios en los parámetros disponibles en las diferentes pestañas.

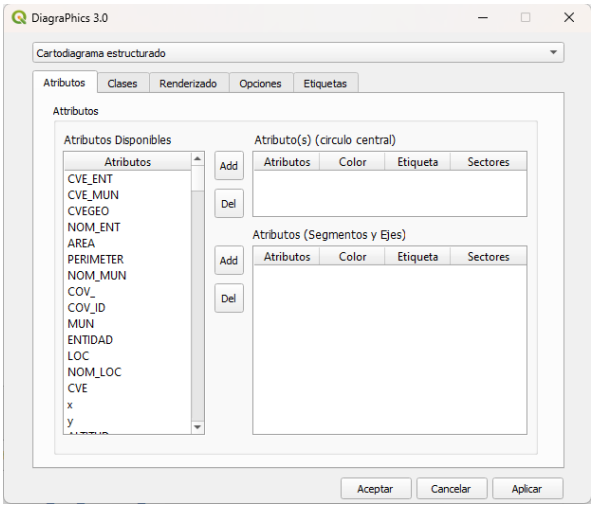

Figura 10 Ventana principal

3.4.2.1.1 Pestaña Atributos.

En esta pestaña se puede seleccionar, agregar o eliminar y cambiar el color de relleno de los atributos disponibles para generar los cartodiagramas y tipogramas.

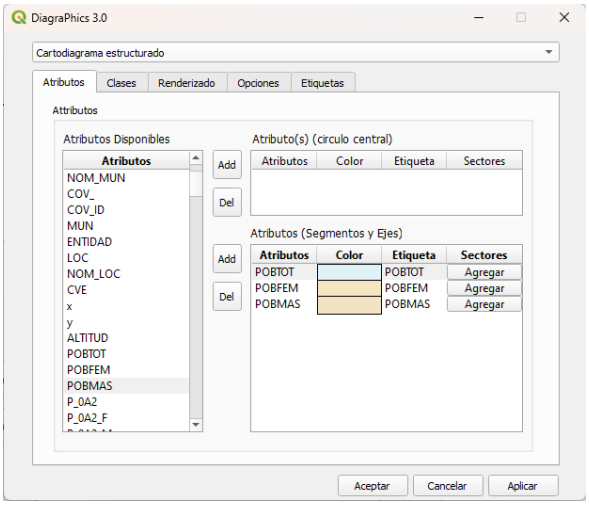

Figura 11 Pestaña atributos

3.4.2.1.2 Pestaña Clases.

En esta pestaña se puede ver, editar o agregar las clases para la agrupación de los datos que se usan para generar los cartodiagramas o tipogramas.

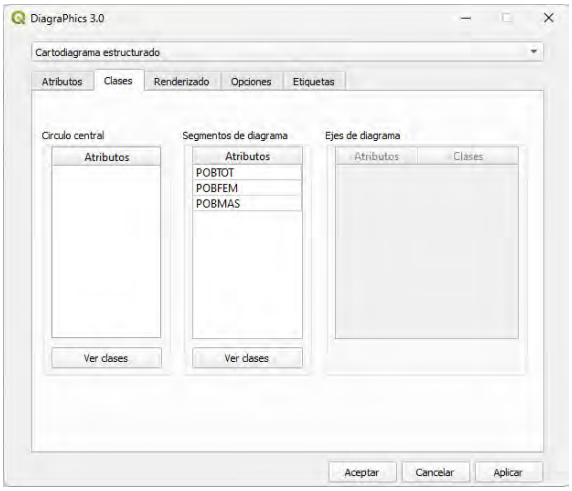

Figura 12 Pestaña clases

3.4.2.1.3 Pestaña Renderizado.

En esta pestaña se puede el color y ancho de línea, la opacidad, el tamaño proporcional, el Angulo y el tamaño de la separación entre sectores de los cartodiagramas y tipogramas.

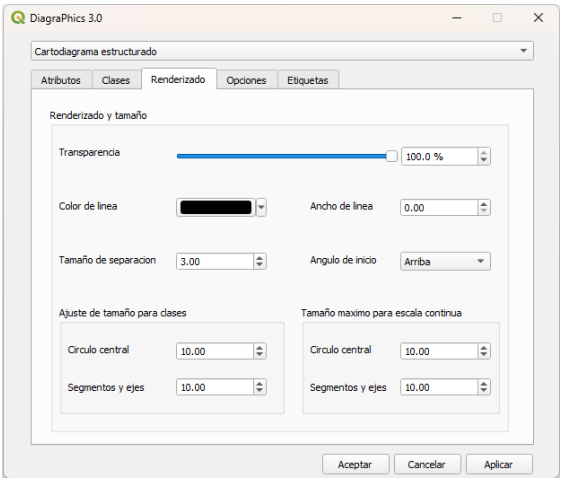

Figura 13 Pestaña renderizado

3.4.2.1.4 Ventana agregar sectores.

En esta ventana se puede agregar sectores a los segmentos que se agregan con los atributos que se agregan en la pestaña atributos Figura 11 de la ventana inicial.Figura 10 Ventana principal

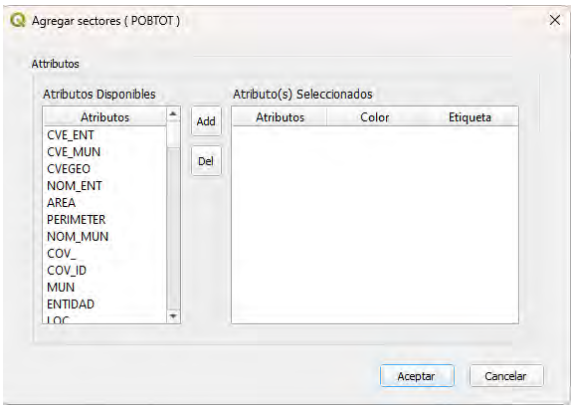

Figura 14 Ventana agregar sectores

3.4.2.1.5 Ventana Clases.

En esta ventana se puede ver, agregar, modificar, eliminar y cambiar el método mediante el cual se generan las clases automáticamente, así como activar la opción usar los datos en escala continua y omitir el uso de las clases.

Se puede acceder a esta ventana desde los botones ver clases en la pestaña clases de la ventana principal. Figura 13.

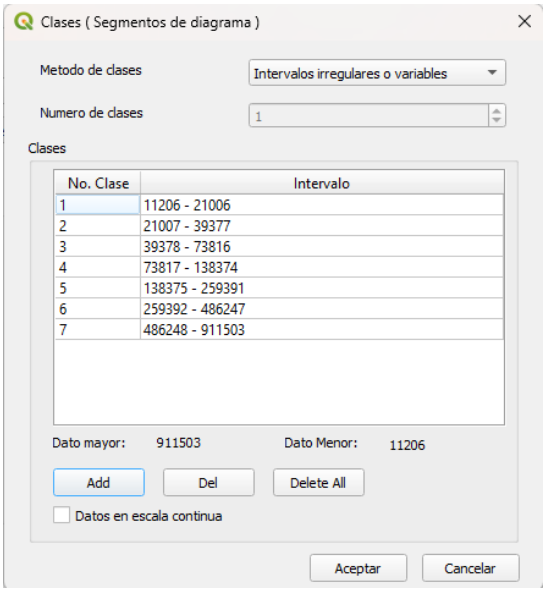

Figura 15 ventana clases

3.4.3 Evaluación.

3.4.3.1 Uso del prototipo.

Para probar el funcionamiento se hace usó de un caso práctico con datos reales, que permitirá observar las capacidades del plug-in para la construcción de cartodiagramas y tipogramas.

El plug-in tiene la capacidad para representar una gran variedad de datos de distintas temáticas, en este caso se usaron datos del Censo de Población y Vivienda, (Iter, INEGI), 2020 en formato Shape de ESRI vectorial poligonal de los límites municipales del estado de Quintana Roo.

En la se describen los atributos del archivo en formato Shape que se usaron para generar los diferentes mapas de ejemplo.

Tabla 31 Descripción de atributos

Nombre del atributo	Descripción
VIVTOT	Total de viviendas
POBFEM	Población femenina
POBMAS	Población Masculina
VPH_INTER	Viviendas particulares habitadas que disponen de Internet
VPH_SINCIN	Viviendas particulares habitadas sin computadora ni Internet
P_0A2	Población de 0 a 2 años
P_3A5	Población de 3 a 5 años
P_6A11	Población de 6 a 11 años
P_12A14	Población de 12 a 14 años
P_15A17	Población de 15 a 17 años
VPH_RADIO	Viviendas particulares habitadas que disponen de radio
VPH_TV	Viviendas particulares habitadas que disponen de televisor
VPH_PC	Viviendas particulares habitadas que disponen de computadora, Tablet o laptop
VPH_TELEF	Viviendas particulares habitadas que disponen de línea telefónica fija
VPH_CEL	Viviendas particulares habitadas que disponen de teléfono celular

3.4.3.1.1 Cartodiagrama de ángulos variables con círculo central simple

En este ejemplo se genera un mapa que muestra el total de viviendas por municipio graduado por tamaño, así como una comparación entre la cantidad de población masculina y femenina, graduado por tamaño de acuerdo al total de población.

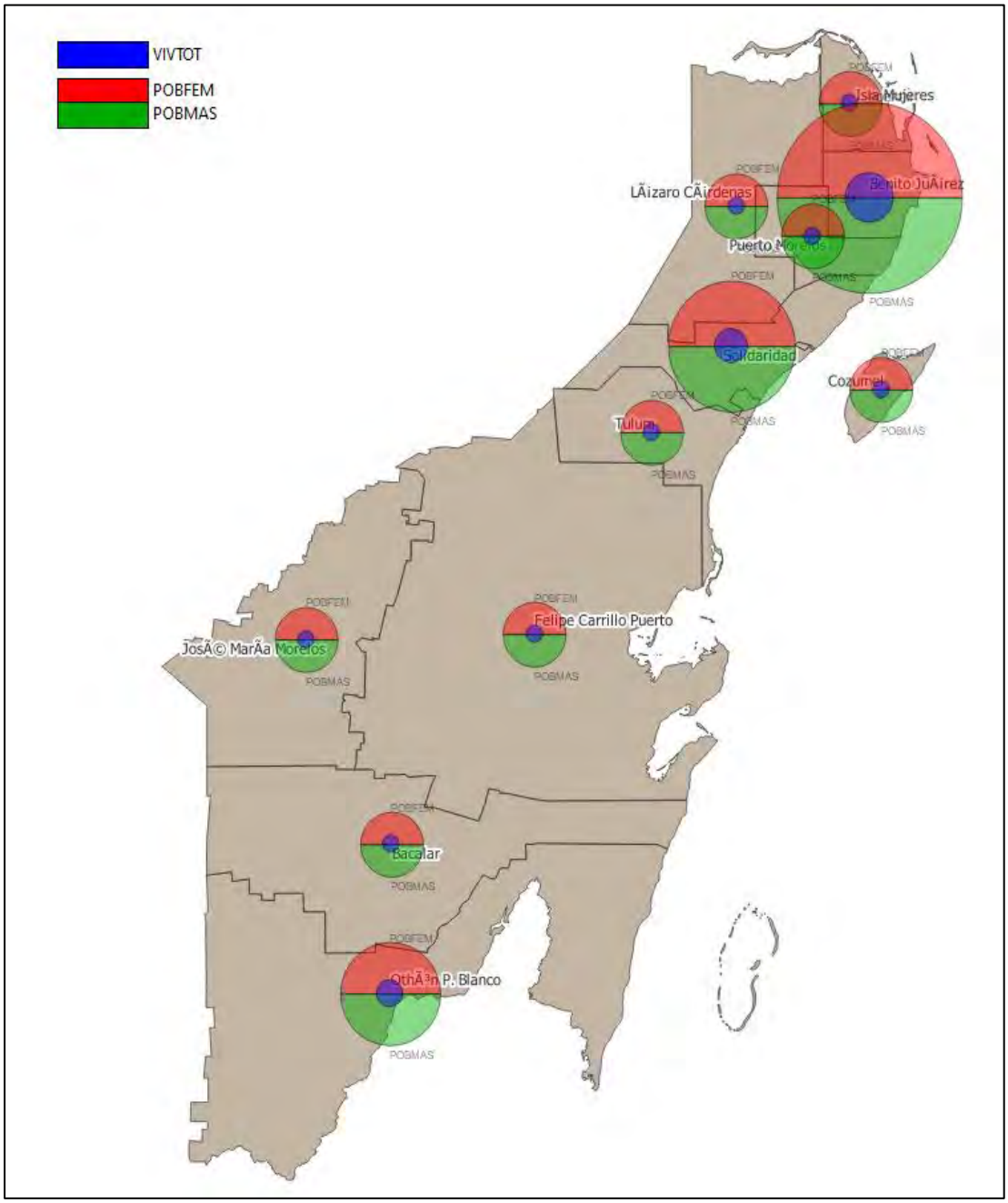

Figura 16 Mapa de cartodiagrama de ángulos variables con círculo central simple

3.4.3.1.2 Cartodiagrama de semicírculos

En este ejemplo se genera un mapa que muestra una comparación entre la cantidad de viviendas con acceso a internet y sin acceso a internet dentro de cada municipio. Así mismo hace la misma comparación entre municipios.

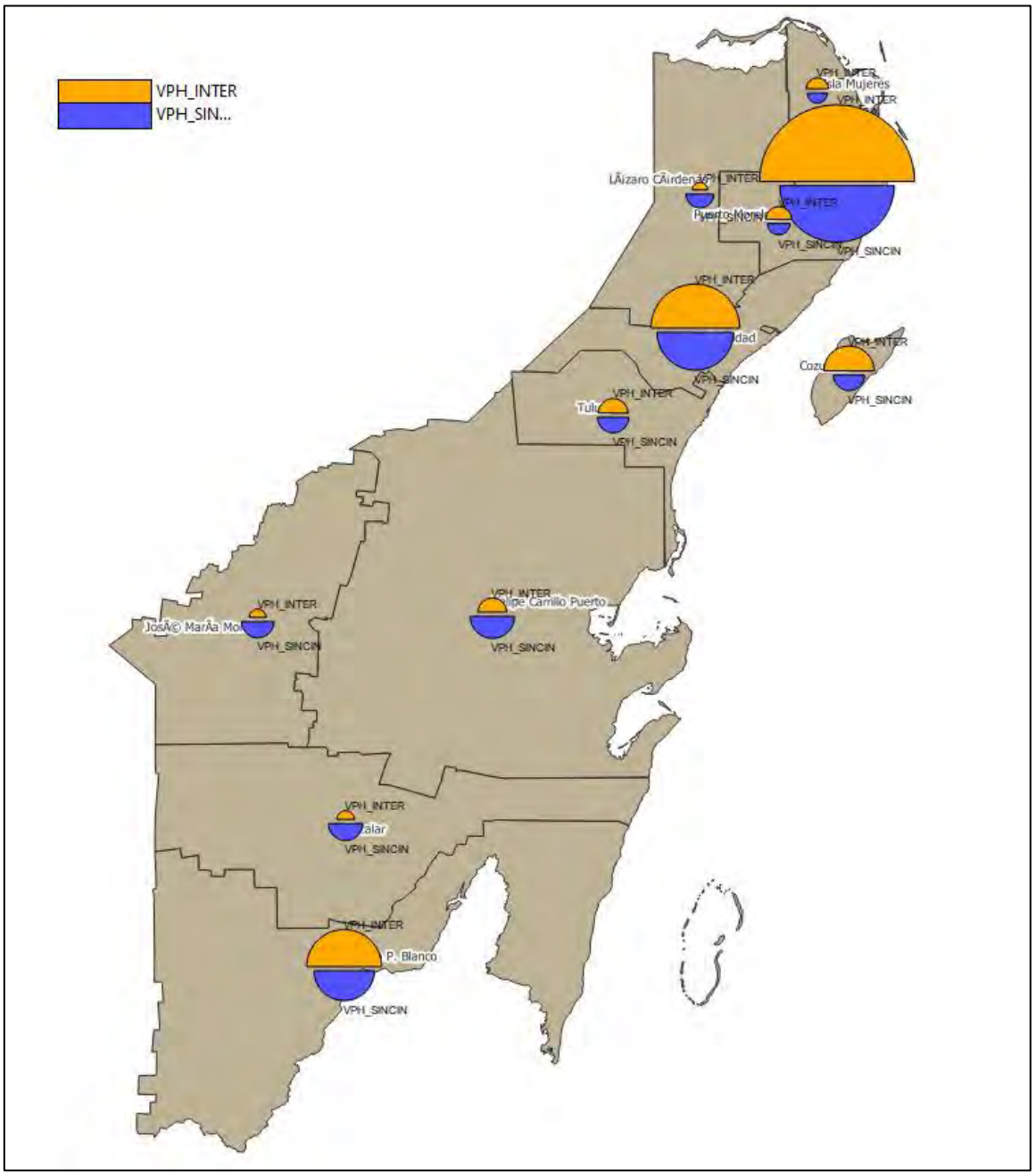

Figura 17 mapa de cartodiagrama de semicírculos

3.4.3.1.3 Tipograma de sectores

En este ejemplo se genera un mapa que muestra los datos de la cantidad menores de edad separadas por rangos de edad en cinco grupos, el mapa nos muestra una comparación entre cada municipio.

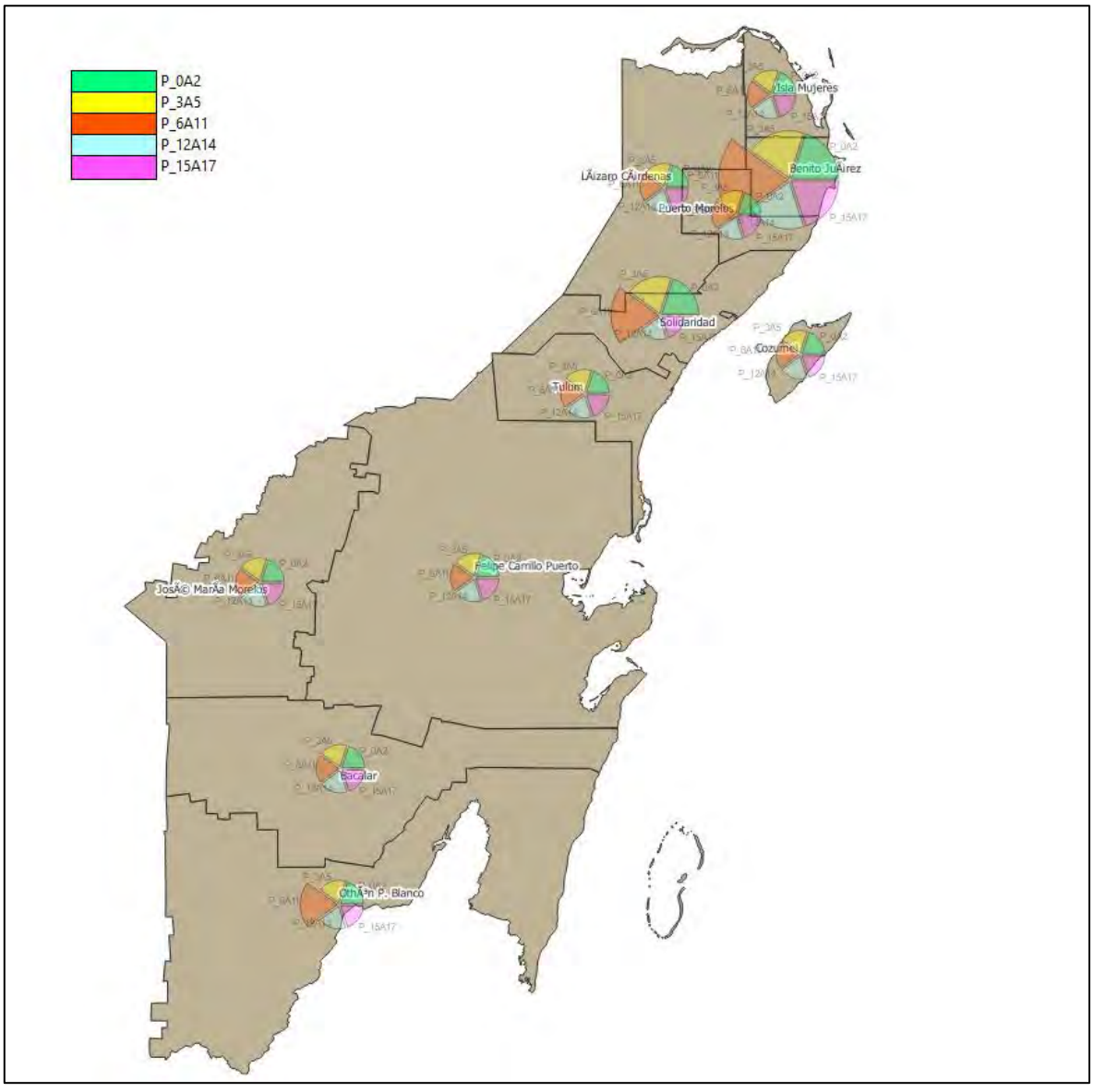

Figura 18 Mapa de tipograma de sectores

3.4.3.1.4 Tipograma de ejes fijos

En este ejemplo se genera un mapa que muestra una comparación entre la cantidad de viviendas que disponen de Radio, Televisión, Computadora, Teléfono y celular por municipio.

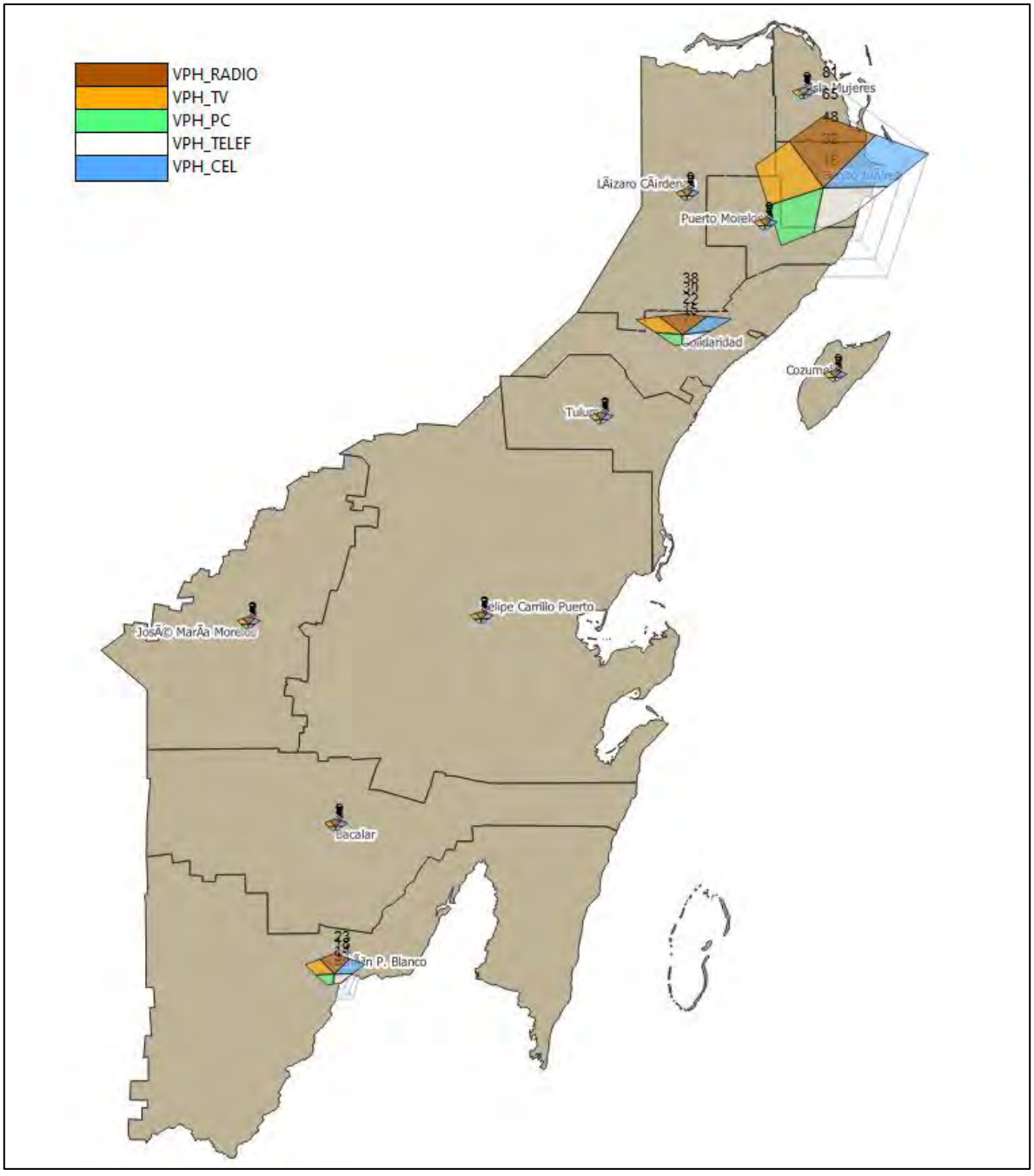

Figura 19 Mapa de tipograma de ejes fijos

CAPÍTULO 4

CONCLUSIONES

CAPÍTULO 4

En el transcurso de esta investigación, se ha desarrollado e implementado un plug-in para la generación de cartodiagramas y tipogramas en QGIS3, con el objetivo de proporcionar una herramienta accesible y eficaz para la representación visual de datos geográficos. Los resultados obtenidos durante este proceso han demostrado la viabilidad y utilidad de la herramienta desarrollada, así como también han destacado áreas de mejora.

Las contribuciones de esta investigación al campo de la visualización geoespacial son significativas. El plug-in desarrollado ofrece una solución práctica y eficiente para la generación de cartodiagramas y tipogramas en QGIS3, lo que permite a los usuarios representar datos geográficos de manera efectiva.

La funcionalidad del plug-in ha sido probada y validada a través de diversos casos de prueba, que incluyen la generación de cartodiagramas y tipogramas utilizando conjuntos de datos geográficos reales. Los resultados de estas pruebas han confirmado la eficacia y la usabilidad del plug-in, destacando su capacidad para crear visualizaciones claras y significativas.

A pesar de los logros alcanzados, es importante reconocer las limitaciones de esta investigación. Entre ellas se incluyen restricciones en la funcionalidad del plug-in, como la falta de ciertas características avanzadas de visualización y la necesidad de mejoras en la interfaz de usuario. Además, se identifican oportunidades para futuras investigaciones, como la exploración de nuevas técnicas de visualización geoespacial y la integración del plug-in en flujos de trabajo geoespaciales más amplios.

En resumen, esta investigación ha demostrado el potencial del desarrollo de herramientas innovadoras para la visualización de datos geográficos en entornos SIG como QGIS3. El plug-in desarrollado representa un paso hacia la mejora de la representación visual de datos espaciales y ofrece nuevas oportunidades para la

exploración y el análisis de información geográfica en una amplia variedad de aplicaciones.

REFERENCIAS BIBLIOGRÁFICAS

Berrios Mena, J. (1992). *Cartografía Digital.* España: Ra-Ma, Libreria y Editorial Microninfomatica.

Cabrero Ortega, M., & García Pérez, A. (2022). *Analísis Estadístico de Datos Espaciales con QGIS y R.* Madrid: UNED.

Caire Lomelí, J. (2002). *Cartografía básica.* Mexico: UNAM, Facultad de Filosofía y Letras.

Escobar, M. d. (2004). *Método y Técnicas de la Cartografía Tematica.* Mexico, D.F.

G, M. E. (20 de Octubre de 2020). *Repositorio digital de la Facultad de Ingeniería - UNAM.* Obtenido de Repositorio digital de la Facultad de Ingeniería - UNAM: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/175/A5%20Capítulo%202.pdf?sequence=5

Gutierrez, A. P. (2016). *Python Paso a Paso.* Madrid: RA-MA.

Instituto Geográfico Agustín Codazzi. (1998). *Principios básicos de cartografía temática.* Colombia: Ministerio de Hacienda y Crédito Público, Instituto Geográfico Agustín Codazzi.

Jimenez, A. M. (2008). *Sistema y Analisis de Informacion Geografica.* Madrid: Editorial RA-MA.

Lawhead, J. (2017). *QGIS Python Programming Cookbook.* Birmingham: Packt Publishing.

Maass, S. F., & Valdez, M. E. (2003). *Principios básicos de cartografía y cartografía automatizada.* Toluca: UAEM.

Martínez Bencardino, C. (2019). *Estadística básica aplicada.* Bogota: Ecoe Ediciones.

McClain, B. P. (2022). *Python for Geospatial Data Analysis.* O'Reilly Media.

Menendez Sanchez, F. J. (2009). *Geodesia Y Cartografia: Los Conceptos y su Aplicacion Practica.* (F. J. Menendez, Ed.) EOSGIS SL. Obtenido de https://books.google.com.mx/books?id=dlUurxOj4A8C&newbks=1&newbks_redir=0&hl=es&redir_esc=y

Menendez Sanchez, F. J. (2009). *Georreferenciación de Cartografia: Datos Raster y Vectoriales.* (F. J. Menendez, Ed.) EOSGIS SL. Obtenido de https://books.google.com.mx/books?id=FWvBWVV2OngC&newbks=1&newbks_redir=0&printsec=frontcover&dq=Georreferenciaci%C3%B3n+de+Cartografia:+Datos+Raster+y+Vectoriales.&hl=es&redir_esc=y#v=onepage&q=Georreferenciaci%C3%B3n%20de%20Cartografia%3A%20Datos%20Rast

Menke, K., Smith, R., Pirelli, L., & Van Hoesen, J. (2016). *Masterin QGIS.* Birmingham: Packt Publishing.

Moreno Jiménez, A., Prieto Flores, M. E., & Martínez Suárez, P. (2008). *Sistemas y Análisis de la Información Geográfica.* España: RA-MA Editorial.

Ocampo Mendieta, J. (2005). *Cartografía básica aplicada.* Colombia: Editorial Universidad de Caldas.

Olaya, V. (2014). *Sistemas de Informacion Geografica.*

OSGEO. (25 de Octubre de 2020). *QGIS*. Obtenido de https://qgis.org/es/site/about/index.html#

Quiróz Hernández, M. (2011). *Tecnologías de la información geográfica (TIG).* España: Universidad de Salamanca.

Quiroz, R. (25 de Octubre de 2020). *gidahatari*. Obtenido de https://gidahatari.com/ih-es/algunos-datos-sobre-pyqgis

Ramos Azcuy, F., & Guerra Bret, R. (2020). *Introducción a los Métodos Estadísticos.* Cuba: Editorial Universitaria.

RedHat, Inc. (28 de Octubre de 2020). *RedHat*. Obtenido de RedHat: https://www.redhat.com/es/devops/what-is-agile-methodology

Riverbank Computing. (25 de Octubre de 2020). *Riverbank Computing*. Obtenido de https://riverbankcomputing.com/software/pyqt/intro

Shammunul Islam, S. M. (2019). *Mastering Geospatial Development with QGIS 3.x.* Birmingham: Packt Publishing.

Sherman, G. (2018). *The PyQGIS Programmer's Guide Extending QGIS 3.x with Python 3.* Reino unido: Locate Press.

Tecnologías de la información geográfica. (2018). Valencia: Universitat de València.

The Qt Company Ltd. (25 de Octubre de 2020). *Qt*. Obtenido de https://doc.qt.io/qt-5/index.html

The Qt Company Ltd. (25 de Octubre de 2020). *Qt*. Obtenido de https://doc.qt.io/qt-5/qtdesigner-manual.html

Vélez Ibarrola, R., Ramos Méndez, E., Carmena Yáñez, V., & Navarro Fernández, J. (2006). *Métodos estadísticos en ciencias sociales.* Ediciones Académicas S.A.

Westra, E. (2014). *Building Mapping Applications with QGIS.* Birmingham: Packt Publishing.

ANEXOS

1.1 Código Fuente

1.1.1 Clase Diagraphics

```
import math
import random
import PyQt5.QtWidgets
from PyQt5.QtWidgets import QMainWindow, QTableWidgetItem
from PyQt5.QtGui import QColor,QPen
from PyQt5.QtCore import QPoint
from qgis.core import QgsSingleCategoryDiagramRenderer,QgsDiagram,QgsDiagramSettings,QgsDiagramLayerSettings
from importlib import reload
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtCore import Qt, pyqtSignal
from PyQt5.QtGui import QColor
from importlib import reload

sys.path.append(r'C:\Users\Victor_CEIQROO\Downloads\QDiagraPhics\QDiagraPhics\Clases')

import Clase_Clase_Box
import Clase_Diagrama_Estructurado
from Clase_Diagram_Render import *
from Clase_Color_Box import *
from Clase_Renderer_Clase import *
import Clase_Btn_Agregar_Atributos_Sectores
from Clase_Segmentos_Diagrama import *
import Clase_Configuracion_Diagrama
import VentanaInicial
import Clase_Ventana_Intervalo_De_Clases
import Clase_Metodos_Calc_Clases
import Clase_Diagrama_Lineal_Dinamico
import Clase_Ejes_Diagrama
import Clase_Ventana_Agregar_Clases
import Clase_Operaciones_Atributos
import Clase_Btn_Agregar_Clases

reload(Clase_Btn_Agregar_Atributos_Sectores)
reload(Clase_Clase_Box)
reload(Clase_Configuracion_Diagrama)
reload(Clase_Diagrama_Estructurado)
reload(Clase_Diagram_Render)
reload(Clase_Metodos_Calc_Clases)
reload(Clase_Diagrama_Lineal_Dinamico)
reload(Clase_Ejes_Diagrama)
reload(Clase_Ventana_Agregar_Clases)
reload(Clase_Operaciones_Atributos)
reload(Clase_Btn_Agregar_Clases)
reload(VentanaInicial)


class DiagraPhics(QMainWindow,VentanaInicial.Ui_MainWindow):
    def __init__(self,lyr,*args,**kwargs):
        QMainWindow.__init__(self,*args,**kwargs)
        self.setupUi(self)
        self.setWindowTitle("DiagraPhics 3.0")
        self.clases=[]
        self.clasesSC=[]
        self.camposSelT1=[]
        self.camposSelT2=[]
        self.camposSelT3=[]
        self.camposSelT4=[]
        self.layer=lyr
        self.listaAtributos=self.layer.fields()
        self.colorLinea.setColor(QColor("black"))
        self.transparencia.setOpacity(1)
        self.ajusteTamT1.setValue(10)
        self.ajusteTamT2.setValue(10)
        self.maximTamT1.setValue(10)
        self.maximTamT2.setValue(10)
        self.distanciaOb.setValue(3)
        self.listarAtributos(self.listaAtributos)
        self.tableClasesED.setEnabled(False)
        self.campos_sel.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.campos_sel.setDragEnabled(False)
```

```
        self.campos_sel_2.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.campos_sel_2.setDragEnabled(False)
        self.campos_capa.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.campos_capa.setDragEnabled(False)
        self.tableClasesCC.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.tableClasesCC.setDragEnabled(False)
        self.tableClasesSC.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.tableClasesSC.setDragEnabled(False)
        self.tableClasesED.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.tableClasesED.setDragEnabled(False)


        self.metodoClases=Clase_Metodos_Calc_Clases.MetodosCalcClases()


        self.button_Add.clicked.connect(self.agregarAtributo)
        self.button_Add_2.clicked.connect(self.agregarAtributo)

        self.buttonDel.clicked.connect(self.quitarAtributo)
        self.buttonDel_2.clicked.connect(self.quitarAtributo)

        self.buttonAceptar.clicked.connect(self.dibujarDiagrama)

        self.diagramSel.activated.connect(self.cambiarDiagrama)


    def cambiarDiagrama(self):
        if self.diagramSel.currentIndex()==0:
            self.tableClasesCC.setEnabled(True)
            self.tableClasesSC.setEnabled(True)
            self.button_Add.setEnabled(True)
            self.buttonDel.setEnabled(True)
            self.campos_sel.setEnabled(True)
            self.tableClasesED.setEnabled(False)
            self.btnVerClasesCC.setEnabled(True)
            self.btnVerClasesSC.setEnabled(True)

        elif self.diagramSel.currentIndex()==1:
            self.campos_sel.setEnabled(False)
            self.tableClasesSC.setEnabled(False)
            self.tableClasesCC.setEnabled(False)
            self.tableClasesED.setEnabled(True)
            self.button_Add.setEnabled(False)
            self.buttonDel.setEnabled(False)
            self.btnVerClasesCC.setEnabled(False)
            self.btnVerClasesSC.setEnabled(False)

        elif self.diagramSel.currentIndex()==2:
            self.tableClasesCC.setEnabled(False)
            self.button_Add.setEnabled(False)
            self.buttonDel.setEnabled(False)

        self.actualizarAtributosSelecionados(self.campos_sel_2)

    def actualizarAtributosSelecionados(self,tabAtrib):
        if self.diagramSel.currentIndex()==0:
            self.tableClasesSC.setRowCount(1)
            self.tableClasesSC.removeRow(0)
            self.tableClasesED.setRowCount(1)
            self.tableClasesED.removeRow(0)
            encabezadoItem=QTableWidgetItem()
            encabezadoItem.setText("Sectores")
            tabAtrib.setColumnHidden(3,False)
            tabAtrib.setHorizontalHeaderItem(3,encabezadoItem)
            for n in range(tabAtrib.rowCount()) :
                nombreAtributo=QTableWidgetItem(tabAtrib.item(n,0))
                self.tableClasesSC.setRowCount(self.tableClasesSC.rowCount()+1)
                self.tableClasesSC.setItem(self.tableClasesSC.rowCount()-1,0,QTableWidgetItem(nombreAtributo))
                if tabAtrib.cellWidget(n,3)==None:
                    btnAccion=Clase_Btn_Agregar_Atributos_Sectores.BtnAgregarAtributosSectores(nombreAtributo.text(),self.campos_capa,self.layer,None)
                    self.campos_sel_2.setCellWidget(n,3,btnAccion)
                if n==tabAtrib.rowCount()-1:
                    atributos=self.getAtributosSeleccionados(self.campos_sel_2)
                    self.btnVerClasesSC.ventanaClases.insertarDatos(atributos,list(self.layer.getFeatures()))


        elif self.diagramSel.currentIndex()==1:
            self.tableClasesSC.setRowCount(1)
            self.tableClasesSC.removeRow(0)
            self.tableClasesED.setRowCount(1)
            self.tableClasesED.removeRow(0)
            tabAtrib.setColumnHidden(3,True)
            for n in range(tabAtrib.rowCount()) :
                nombreAtributo=QTableWidgetItem(tabAtrib.item(n,0))
                self.tableClasesED.setRowCount(self.tableClasesED.rowCount()+1)
                btnAccion=Clase_Btn_Agregar_Clases.BtnAgregarClases(nombreAtributo.text())
                atributos=self.getAtributoSeleccionados(nombreAtributo.text())
                btnAccion.ventanaClases.insertarDatos(atributos,list(self.layer.getFeatures()))

                self.tableClasesED.setItem(self.tableClasesED.rowCount()-1,0,QTableWidgetItem(nombreAtributo))
                self.tableClasesED.setCellWidget(self.tableClasesED.rowCount()-1,1,btnAccion)
```

```
def compararAtributos(self):
    n=0
    if self.checkCompAtrib.checkState()==Qt.Checked:
        for n in range(self.campos_sel_3.rowCount()):
            colorBox=QColorBox(defaultColor=self.campos_sel_3.cellWidget(n,1).color())
            nuevoItemEtiqueta=QTableWidgetItem(self.campos_sel_3.item(n,2))
            self.campos_sel_4.setCellWidget(n,1,colorBox)
            self.campos_sel_4.setItem(n,2,nuevoItemEtiqueta)
    elif self.checkCompAtrib.checkState()==Qt.Unchecked:
        for n in range(self.campos_sel_3.rowCount()):
            color=[random.randint(180, 255), random.randint(180, 255), random.randint(180, 255)]
            colorBox=QColorBox(defaultColor=QColor(color[0],color[1],color[2]))
            nuevoItemEtiqueta=QTableWidgetItem(self.campos_sel_4.item(n,0))
            self.campos_sel_4.setCellWidget(n,1,colorBox)
            self.campos_sel_4.setItem(n,2,nuevoItemEtiqueta)


def agregarClaseSegmentosCentrales(self):
    clase=Clase_Ventana_Intervalo_De_Clases.VentanaIntervaloClases.setClase()
    if clase:
        nuevaClase=Clase_Clase_Box.claseBox(clase)
        self.tableClasesCC.setRowCount(self.tableClasesCC.rowCount()+1)
        nuevoItemInter=QTableWidgetItem()
        nuevoItemNo=QTableWidgetItem()
        nuevoItemEtiqueta=QTableWidgetItem()
        nuevoItemNo.setText(str(self.tableClasesCC.rowCount()))
        nuevoItemEtiqueta.setText("Editar")
        self.tableClasesCC.setItem(self.tableClasesCC.rowCount()-1,0,nuevoItemNo)
        self.tableClasesCC.setCellWidget(self.tableClasesCC.rowCount()-1,1,nuevaClase)
        self.tableClasesCC.setItem(self.tableClasesCC.rowCount()-1,2,nuevoItemEtiqueta)

def agregarClaseSegementosDeDiagrama(self):
    clase=Clase_Ventana_Intervalo_De_Clases.VentanaIntervaloClases.setClase()
    if clase:
        nuevaClase=Clase_Clase_Box.claseBox(clase)
        self.tableClasesSC.setRowCount(self.tableClasesSC.rowCount()+1)
        nuevoItemInter=QTableWidgetItem()
        nuevoItemNo=QTableWidgetItem()
        nuevoItemEtiqueta=QTableWidgetItem()
        nuevoItemNo.setText(str(self.tableClasesSC.rowCount()))
        nuevoItemEtiqueta.setText("Editar")
        self.tableClasesSC.setItem(self.tableClasesSC.rowCount()-1,0,nuevoItemNo)
        self.tableClasesSC.setCellWidget(self.tableClasesSC.rowCount()-1,1,nuevaClase)
        self.tableClasesSC.setItem(self.tableClasesSC.rowCount()-1,2,nuevoItemEtiqueta)


def agregarClasesTabla(self,tablaClases,clases):
    self.deleteAllClases(tablaClases)
    for clase in clases:
        nuevaClase=Clase_Clase_Box.claseBox(clase)
        tablaClases.setRowCount(tablaClases.rowCount()+1)
        inter=str(clase.min)+"-"+str(clase.max)
        nuevoItemInter=QTableWidgetItem()
        nuevoItemNo=QTableWidgetItem()
        nuevoItemEtiqueta=QTableWidgetItem()
        nuevoItemNo.setText(str(tablaClases.rowCount()))
        nuevoItemInter.setText(inter)
        nuevoItemEtiqueta.setText("Editar")
        tablaClases.setItem(tablaClases.rowCount()-1,0,nuevoItemNo)
        tablaClases.setCellWidget(tablaClases.rowCount()-1,1,nuevaClase)
        tablaClases.setItem(tablaClases.rowCount()-1,2,nuevoItemEtiqueta)



def cargarDatosAtributos(self,atributo):
    feat=list(self.layer.getFeatures())
    datosAtributo=[]
    for f in feat:
        datosAtributo.append(f[atributo.name()])
    return datosAtributo

def listarAtributos(self,fieldList):
    for atributo in fieldList:
        self.campos_capa.setRowCount(self.campos_capa.rowCount()+1)
        atributoItem=QTableWidgetItem()
        atributoItem.setText(atributo.name())
        self.campos_capa.setItem(self.campos_capa.rowCount()-1,0,atributoItem)


def agregarAtributo(self):
    if self.campos_capa.selectedItems() :
        nameSender=self.sender().objectName()
        color=[random.randint(180, 255), random.randint(180, 255), random.randint(180, 255)]
        nuevoColorBox=colorBox(defaultColor=QColor(color[0],color[1],color[2]))


        if not Clase_Operaciones_Atributos.validarAtributo(self.campos_capa.selectedItems()[0].text(),self.layer.getFeatures()):
```

```
            QMessageBox.critical(iface.mainWindow(), "Error de datos", 'El atributo seleccionado contiene datos invalidos')
            return

        if nameSender=="button_Add" :
            self.campos_sel.setRowCount(self.campos_sel.rowCount()+1)
            self.tableClasesCC.setRowCount(self.tableClasesCC.rowCount()+1)
            nombreAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])
            btnAgregarSectores=Clase_Btn_Agregar_Atributos_Sectores.BtnAgregarAtributosSectores(nombreAtributo.text(),self.campos_capa,None)
            etiquetaAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,0,nombreAtributo)
            self.campos_sel.setCellWidget(self.campos_sel.rowCount()-1,1,nuevoColorBox)
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,2,etiquetaAtributo)
            self.campos_sel.setCellWidget(self.campos_sel.rowCount()-1,3,btnAgregarSectores)
            self.tableClasesCC.setItem(self.tableClasesCC.rowCount()-1,0,QTableWidgetItem(nombreAtributo))
            atributos=self.getAtributosSeleccionados(self.campos_sel)
            self.btnVerClasesCC.ventanaClases.insertarDatos(atributos,list(self.layer.getFeatures()))


        elif nameSender=="button_Add_2" :
            self.campos_sel_2.setRowCount(self.campos_sel_2.rowCount()+1)
            nombreAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])

            etiquetaAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])

            self.campos_sel_2.setItem(self.campos_sel_2.rowCount()-1,0,nombreAtributo)
            self.campos_sel_2.setCellWidget(self.campos_sel_2.rowCount()-1,1,nuevoColorBox)
            self.campos_sel_2.setItem(self.campos_sel_2.rowCount()-1,2,etiquetaAtributo)

            if self.diagramSel.currentIndex()==0:
                self.tableClasesSC.setRowCount(self.tableClasesSC.rowCount()+1)
                btnAccion=Clase_Btn_Agregar_Atributos_Sectores.BtnAgregarAtributosSectores(nombreAtributo.text(),self.campos_capa,self.layer,self)
                self.tableClasesSC.setItem(self.tableClasesSC.rowCount()-1,0,QTableWidgetItem(nombreAtributo))
                self.campos_sel_2.setCellWidget(self.campos_sel_2.rowCount()-1,3,btnAccion)
                atributos=self.getAtributosSeleccionados(self.campos_sel_2)
                self.btnVerClasesSC.ventanaClases.insertarDatos(atributos,list(self.layer.getFeatures()))



            elif self.diagramSel.currentIndex()==1:

                self.tableClasesED.setRowCount(self.tableClasesED.rowCount()+1)
                atributos=self.getAtributoSeleccionados(nombreAtributo.text())
                btnAccion=Clase_Btn_Agregar_Clases.BtnAgregarClases(nombreAtributo.text())
                btnAccion.ventanaClases.insertarDatos(atributos,list(self.layer.getFeatures()))

                self.tableClasesED.setItem(self.tableClasesED.rowCount()-1,0,QTableWidgetItem(nombreAtributo))
                self.tableClasesED.setCellWidget(self.tableClasesED.rowCount()-1,1,btnAccion)


            if self.campos_sel_2.rowCount()>=2 :
                pass
                #self.agregarClasesTabla(self.tableClasesSC,self.crearClases(self.campos_sel_2))
                #self.agregarClasesTabla2(self.tableClasesSC,self.campos_sel_2)


    def quitarAtributo(self):
        nameSender=self.sender().objectName()
        if nameSender=="buttonDel" :
            if self.campos_sel.selectedItems() :
                self.tableClasesCC.removeRow(self.campos_sel.currentRow())
                self.campos_sel.removeRow(self.campos_sel.currentRow())
                #self.deleteAllClases(self.tableClasesCC)
                #self.agregarClasesTabla(self.tableClasesCC,self.crearClases(self.campos_sel))


        elif nameSender=="buttonDel_2" :
            if self.campos_sel_2.selectedItems() :
                if self.diagramSel.currentIndex()==0:
                    self.tableClasesSC.removeRow(self.campos_sel_2.currentRow())
                elif self.diagramSel.currentIndex()==1:
                    self.tableClasesED.removeRow(self.campos_sel_2.currentRow())
                self.campos_sel_2.removeRow(self.campos_sel_2.currentRow())
                if self.campos_sel_2.rowCount()>=2 :
                    pass



    def getPartesDiagrama(self,*tabAtrib):
        partesDiagrama=[]
        for tab in tabAtrib :
            for n in range(tab.rowCount()) :
                for atributo in self.listaAtributos:
                    if atributo.name()==tab.item(n,0).text():
                        atributoTemp=atributo


            datos=self.cargarDatosAtributos(atributoTemp)
            if self.diagramSel.currentIndex()==0:
```

```
                parteNueva=SegmentosDiagrama(n,tab.cellWidget(n,1).color(),tab.item(n,2).text(),atributoTemp,datos,tab.cellWidget(n,3).ventanaSectores.atributos)

            if self.diagramSel.currentIndex()==1:

parteNueva=Clase_Ejes_Diagrama.EjesDiagrama(n,tab.cellWidget(n,1).color(),tab.item(n,2).text(),atributoTemp,datos,self.tableClasesED.cellWidget(n,1).ventanaCla
ses.clases,self.tableClasesED.cellWidget(n,1).ventanaClases.checkEscContCC.checkState())
            else:
                parteNueva=SegmentosDiagrama(n,tab.cellWidget(n,1).color(),tab.item(n,2).text(),atributoTemp,datos,tab.cellWidget(n,3).ventanaSectores.atributos)


            partesDiagrama.append(parteNueva)
        return partesDiagrama

    def getAtributosSeleccionados(self,*tabAtrib):
        atributosSelecionados=[]
        for tab in tabAtrib :
            for n in range(tab.rowCount()) :
                for atributo in self.listaAtributos:
                    if atributo.name()==tab.item(n,0).text():
                        atributosSelecionados.append(atributo)
        return atributosSelecionados

    def getAtributoSeleccionados(self,atributo):
        atributosSelecionados=[]
        for atrib in self.listaAtributos:
            if atrib.name()==atributo:
                atributosSelecionados.append(atrib)
        return atributosSelecionados


    def dibujarDiagrama(self):
        if self.diagramSel.currentIndex()==0:
            diagram=Clase_Diagrama_Estructurado.diagramaEstructurado()


        elif self.diagramSel.currentIndex()==1 :
            diagram=Clase_Diagrama_Lineal_Dinamico.diagramaLinealDinamico()

        else:
            pass


        segmentos=self.getPartesDiagrama(self.campos_sel,self.campos_sel_2)
        atributos=[]
        colors=[]
        etiquetas=[]

        for attrib in segmentos:
            atributos.append(attrib.campo.name())
            if hasattr(attrib,'atributosSectores'):
                for atribSectores in  attrib.atributosSectores:
                    atributos.append(atribSectores.nombre)
            colors.append(attrib.color)
            etiquetas.append(attrib.etiqueta)

        ds=QgsDiagramSettings()
        ds.categoryColors = colors
        ds.categoryAttributes = atributos
        ds.categoryLabels = etiquetas
        ds.penColor=QColor(self.colorLinea.color())
        ds.opacity=self.transparencia.opacity()
        ds.penWidth=self.anchoLinea.value()
        ds.minimumSize=self.ajusteTamT1.value()
        ds.barWidth=self.distanciaOb.value()


        sd=Clase_Configuracion_Diagrama.configuracionDiagrama()
        sd.penColor=self.colorLinea.color()
        sd.opacity=self.transparencia.opacity()
        sd.penWidth=self.anchoLinea.value()
        sd.fixedSizeT1=self.ajusteTamT1.value()
        sd.fixedSizeT2=self.ajusteTamT2.value()
        sd.maximunSizeT1=self.maximTamT1.value()
        sd.maximunSizeT2=self.maximTamT2.value()
        sd.distancia=self.distanciaOb.value()

        if self.diagramSel.currentIndex()==0:
            sd.setClasesSegmentosCentrales(self.btnVerClasesCC.ventanaClases.clases)
            sd.setClasesSegmentos(self.btnVerClasesSC.ventanaClases.clases)
            sd.escalaContinuaT1=self.btnVerClasesCC.ventanaClases.checkEscContCC.checkState()
            sd.escalaContinuaT2=self.btnVerClasesSC.ventanaClases.checkEscContCC.checkState()
            sd.segmentosCentrales=self.getPartesDiagrama(self.campos_sel)
            sd.segmentosDeDiagrama=self.getPartesDiagrama(self.campos_sel_2)
        elif self.diagramSel.currentIndex()==1:
            sd.segmentosDeDiagrama=self.getPartesDiagrama(self.campos_sel_2)

        dr=diagramRender()
        dr.setDiagram(diagram)
        dr.setDiagramSettings(ds)
        dr.setSettingsDiagram(sd)
        dls=QgsDiagramLayerSettings()
```

```
        self.layer.setDiagramRenderer(dr)
        self.layer.setDiagramLayerSettings(dls)

        self.layer.triggerRepaint()


layer=iface.activeLayer()
app = DiagraPhics(layer)
app.show()
```

1.1.2 Clase diagramRender

```
import Clase_Configuracion_Diagrama
from qgis.core import QgsLinearlyInterpolatedDiagramRenderer,QgsDiagram,QgsDiagramSettings,QgsDiagramLayerSettings

class diagramRender(QgsLinearlyInterpolatedDiagramRenderer):
    def __init__(self,*args):
        super().__init__()
        self.cont=1
        self.mSettings=None
        self.mSettingsDiagram=Clase_Configuracion_Diagrama.configuracionDiagrama()
        self.mShowAttributeLegend=True
        self.d=None
        if args :
            self.d=args[0].diagram().clone()
            self.setDiagram(self.d)
            self.mSettings=args[0].mSettings
            self.mSettingsDiagram=args[0].mSettingsDiagram
            self.setDiagramSettings(self.mSettings)

    def clone(self):
        return diagramRender(self)

    def setDiagramSettings(self,s):
        self.mSettings=s

    def diagramSettingsTwo(self,feature,c,s):
        s=self.mSettings
        s.size = self.diagramSize(feature,c)
        return True

    def sizeMapUnits(self,feature,c):
        s = QgsDiagramSettings();
        if ( not self.diagramSettingsTwo(feature,c,s)):
            return QSizeF()
        size = self.diagramSize(feature,c)
        if (size.isValid()):
            width = c.convertToMapUnits(size.width(),s.sizeType,s.sizeScale)
            size.setHeight((size.height())*(width / size.width()))
            size.setWidth(width)
        self.cont=self.cont+1
        return size;

    def diagramSize(self,feature,c):
        return self.d.diagramSize(feature,c,self.mSettings)

    def diagramAttributes(self):
        return self.mSettings.categoryAttributes

    def setFieldSel(self,datos):
        self.diagram().setCamposSel(datos)

    def fieldSel(self):
        return self.mSettingsDiagram.fieldSel

    def setDataFields(self,dataFields):
        self.diagram().setDataFieldDiagram(dataFields)

    def setSettingsDiagram(self,sd):
        self.diagram().setSettingsDiagram(sd)

    def settingsDiagram(self):
        return self.mSettingsDiagram
```

1.1.3 Clase DiagramaLinealDinamico

```
import math
from importlib import reload
import Clase_Configuracion_Diagrama
from PyQt5.QtGui import *
from qgis.core import QgsDiagram
from PyQt5.QtCore import *
```

```
reload(Clase_Configuracion_Diagrama)
class diagramaLinealDinamico(QgsDiagram):
    def __init__(self,*args):
        super().__init__()
        self.mSettingsDiagram=Clase_Configuracion_Diagrama.configuracionDiagrama()
        self.mPen=QPen()
        if args :
            self.mSettingsDiagram=args[0].mSettingsDiagram
            self.mPen=args[0].mPen

    def clone(self):
        return diagramaLinealDinamico(self)
        yelf.mSettingsDiagram=sd

    def setSettingsDiagram(self,sd):
        self.mSettingsDiagram=sd


    def diagramName(self):
        return "Tipograma Lineal Dinamico"

    def diagramSize(self,feature,c,s):
        size=QSizeF(1,1)
        return size

    def renderDiagram(self,feature,c,s,p):
        posX=p.x()
        posY=p.y()
        pos=QPoint(posX,posY)
        dist=self.mSettingsDiagram.distancia
        fixedSizeT1=self.mSettingsDiagram.fixedSizeT1
        fixedSizeT2=self.mSettingsDiagram.fixedSizeT2
        maximunSizeT1=self.mSettingsDiagram.maximunSizeT1
        maximunSizeT2=self.mSettingsDiagram.maximunSizeT2
        longMax=self.mSettingsDiagram.maximunSizeT2

        spu = self.sizePainterUnits(s.size,s,c)
        self.setPenWidth(self.mPen,s,c)
        angulo=360/len(self.mSettingsDiagram.segmentosDeDiagrama)
        anguloSig=0
        polygon=QPolygonF()
        vertices=[]
        colores=[]
        simbolo=c.painter()
        simbolo.setPen(self.mPen)
        maximSuma = 0
        for n in self.mSettingsDiagram.segmentosDeDiagrama:
            maximSuma = maximSuma + n.getMax();

        maxim = maximSuma/len(self.mSettingsDiagram.segmentosDeDiagrama)

        maxsize = 0
        for n in self.mSettingsDiagram.segmentosDeDiagrama:
            value=int(feature[n.campo.name()])
            if n.getEscalaContinua():
                tam=((maximunSizeT2*value)/maxim)
            else:
                clases=n.getClases()
                if clases:
                    escala=fixedSizeT2
                    for clase in clases:
                        if value >= clase.min and value <= clase.max :
                            tam=((fixedSizeT2*clase.getMed())/maxim)
                            break


            if tam > maxsize :
                maxsize = tam

            if anguloSig==0:
                PAX=posX
                PAY=posY-tam


            if anguloSig>0 and anguloSig<=90:
                anguloA=(90-anguloSig)
                anguloC=180-(90+anguloA)
                ladoA=(math.sin(math.radians(anguloC)))*(tam/math.sin(math.radians(90)))
                ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
                PAX=posX-ladoA
                PAY=posY-ladoB


            if anguloSig>90 and anguloSig<=180:
                anguloA=(anguloSig-90)
                anguloC=180-(90+anguloA)
                ladoA=(math.sin(math.radians(anguloC)))*(tam/math.sin(math.radians(90)))

                if anguloC != 0:
                    ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
                else:
```

```
            ladoB=tam
        PAX=posX-ladoA
        PAY=posY+ladoB


    if anguloSig>180 and anguloSig<=270:
        anguloA=(270-anguloSig)
        anguloC=180-(90+anguloA)
        ladoA=(math.sin(math.radians(anguloC)))*(tam/math.sin(math.radians(90)))
        ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
        PAX=posX+ladoA
        PAY=posY+ladoB


    if anguloSig>270 and anguloSig<=360:
        anguloA=(anguloSig-270)
        anguloC=180-(90+anguloA)
        ladoA=(math.sin(math.radians(anguloC)))*(tam/math.sin(math.radians(90)))
        ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
        PAX=posX+ladoA
        PAY=posY-ladoB

    vertices.append(QPointF(PAX,PAY))
    colores.append(n.color)
    anguloSig=anguloSig+angulo


anguloSig = 0
verticesBase=[]
font = QFont('Arial', 6)
simbolo.setFont(font)
metrics = simbolo.fontMetrics()
textWidth = 0
for n in self.mSettingsDiagram.segmentosDeDiagrama:
    if anguloSig==0:
        PAX=posX
        PAY=posY-maxsize
        textWidth = metrics.horizontalAdvance(n.campo.name())


    if anguloSig>0 and anguloSig<=90:
        anguloA=(90-anguloSig)
        anguloC=180-(90+anguloA)
        ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))
        ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
        PAX=posX-ladoA
        PAY=posY-ladoB
        textWidth = metrics.horizontalAdvance(n.campo.name())


    if anguloSig>90 and anguloSig<=180:
        anguloA=(anguloSig-90)
        anguloC=180-(90+anguloA)
        ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))

        if anguloC != 0:
            ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
        else:
            ladoB=maxsize
        PAX=posX-ladoA
        PAY=posY+ladoB
        textWidth = metrics.horizontalAdvance(n.campo.name())


    if anguloSig>180 and anguloSig<=270:
        anguloA=(270-anguloSig)
        anguloC=180-(90+anguloA)
        ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))
        ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
        PAX=posX+ladoA
        PAY=posY+ladoB
        textWidth = 0



    if anguloSig>270 and anguloSig<=360:
        anguloA=(anguloSig-270)
        anguloC=180-(90+anguloA)
        ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))
        ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
        PAX=posX+ladoA
        PAY=posY-ladoB
        textWidth = 0


    verticesBase.append([QPointF(PAX,PAY),n.campo.name(),textWidth])

    anguloSig=anguloSig+angulo

for vertice in verticesBase:
```

```
    self.mPen.setColor(QColor("#bcbcbc"))
    simbolo.setPen(self.mPen)
    simbolo.drawLine(posX,posY,vertice[0].x(),vertice[0].y())
    self.mPen.setColor(self.mSettingsDiagram.penColor)
    simbolo.setPen(self.mPen)
    #simbolo.drawText(vertice[0].x()-vertice[2],vertice[0].y(), vertice[1])


numTrazos = 5
widthTrazo = maxsize/numTrazos
maxsize = 0
font = QFont('Arial', 8)
simbolo.setFont(font)
for i in range(numTrazos):
    maxsize = maxsize + widthTrazo
    anguloSig = 0
    verticesTrazo = []
    for n in self.mSettingsDiagram.segmentosDeDiagrama:

        if anguloSig==0:
            PAX=posX
            PAY=posY-maxsize
            self.mPen.setColor(self.mSettingsDiagram.penColor)
            simbolo.setPen(self.mPen)
            simbolo.drawText(QPointF(PAX, PAY), str(int(maxsize)))


        if anguloSig>0 and anguloSig<=90:
            anguloA=(90-anguloSig)
            anguloC=180-(90+anguloA)
            ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))
            ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
            PAX=posX-ladoA
            PAY=posY-ladoB


        if anguloSig>90 and anguloSig<=180:
            anguloA=(anguloSig-90)
            anguloC=180-(90+anguloA)
            ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))

            if anguloC != 0:
                ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
            else:
                ladoB=maxsize
            PAX=posX-ladoA
            PAY=posY+ladoB


        if anguloSig>180 and anguloSig<=270:
            anguloA=(270-anguloSig)
            anguloC=180-(90+anguloA)
            ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))
            ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
            PAX=posX+ladoA
            PAY=posY+ladoB



        if anguloSig>270 and anguloSig<=360:
            anguloA=(anguloSig-270)
            anguloC=180-(90+anguloA)
            ladoA=(math.sin(math.radians(anguloC)))*(maxsize/math.sin(math.radians(90)))
            ladoB=(math.sin(math.radians(anguloA)))*(ladoA/math.sin(math.radians(anguloC)))
            PAX=posX+ladoA
            PAY=posY-ladoB

        verticesTrazo.append(QPointF(PAX,PAY))
        anguloSig=anguloSig+angulo

    simbolo.setOpacity(1)
    self.mPen.setColor(QColor("#bcbcbc"))
    simbolo.setPen(self.mPen)
    for j in range(len(verticesTrazo)):
        if j < (len(verticesTrazo) - 1):
            simbolo.drawLine(verticesTrazo[j].x(),verticesTrazo[j].y(),verticesTrazo[j+1].x(),verticesTrazo[j+1].y())
        else:
            simbolo.drawLine(verticesTrazo[j].x(),verticesTrazo[j].y(),verticesTrazo[0].x(),verticesTrazo[0].y())

poligonos=[]
self.mPen.setColor(self.mSettingsDiagram.penColor)
simbolo.setPen(self.mPen)
simbolo.setOpacity(self.mSettingsDiagram.opacity-0.2)
for n in range(len(vertices)):
    path = QPainterPath()
    simbolo.setBrush(colores[n])
    if n!=0:
        xi=vertices[n-1].x()
        yi=vertices[n-1].y()
    else:
        xi=vertices[len(vertices)-1].x()
```

```
            yi=vertices[len(vertices)-1].y()

        xc=vertices[n].x()
        yc=vertices[n].y()

        if n!= len(vertices)-1:
            xf=vertices[n+1].x()
            yf=vertices[n+1].y()
        else:
            xf=vertices[0].x()
            yf=vertices[0].y()

        XMI=(xc+xf)/2
        YMI=(yc+yf)/2
        XMD=(xc+xi)/2
        YMD=(yc+yi)/2

        polygon=QPolygonF()
        polygon.append(QPointF(posX,posY))
        polygon.append(QPointF(XMI,YMI))
        polygon.append(QPointF(xc,yc))
        polygon.append(QPointF(XMD,YMD))
        polygon.append(QPointF(posX,posY))
        path.addPolygon(polygon)
        simbolo.drawPath(path)
```

1.1.4 Clase DiagramaEstructurado

```
import math
from importlib import reload
import Clase_Configuracion_Diagrama
from PyQt5.QtGui import *
from qgis.core import QgsDiagram
from PyQt5.QtCore import *

reload(Clase_Configuracion_Diagrama)
class diagramaEstructurado(QgsDiagram):
    def __init__(self,*args):
        super().__init__()

        self.mSettingsDiagram=Clase_Configuracion_Diagrama.configuracionDiagrama()
        self.mPen=QPen()
        if args :
            self.mSettingsDiagram=args[0].mSettingsDiagram
            self.mPen=args[0].mPen

    def clone(self):
        return diagramaEstructurado(self)

    def setSettingsDiagram(self,sd):
        self.mSettingsDiagram=sd

    def diagramName(self):
        return "Cartodiagrama Estructurado"

    def diagramSize(self,feature,c,s):
        size=QSizeF(1,1)
        return size

    def renderDiagram(self,feature,c,s,p):
        posX=p.x()
        posY=p.y()
        pos=QPoint(posX,posY)
        dist=self.mSettingsDiagram.distancia + 0.1
        fixedSizeT1=self.mSettingsDiagram.fixedSizeT1
        fixedSizeT2=self.mSettingsDiagram.fixedSizeT2
        maximunSizeT1=self.mSettingsDiagram.maximunSizeT1
        maximunSizeT2=self.mSettingsDiagram.maximunSizeT2
        tam=1
        spu = self.sizePainterUnits(s.size,s,c)
        #spu= QSizeF(1,1)
        self.mPen.setColor(self.mSettingsDiagram.penColor)
        self.setPenWidth(self.mPen,s,c)
        maxim=self.mSettingsDiagram.maxValSegmentosDeDiagrama()
        minim=self.mSettingsDiagram.minValSegmentosDeDiagrama()
        cont=0
        size=0
        anguloInicio=0
        metrics = c.painter().fontMetrics()
        textWidth = 0
        if self.mSettingsDiagram.segmentosDeDiagrama:
            tamAngulo=(360/len(self.mSettingsDiagram.segmentosDeDiagrama))

        for n in self.mSettingsDiagram.segmentosDeDiagrama:
```

```
    value=int(feature[n.campo.name()])
    if self.mSettingsDiagram.escalaContinuaT2:
        ajusTam=((maximunSizeT2/math.sqrt(maxim))*2)
        size=((spu.width()*(math.sqrt(value)*ajusTam))*tam)
    else:
        clases=self.mSettingsDiagram.getClasesSegmentos()
        if clases:
            escala=fixedSizeT2
            for clase in clases:
                if value >= clase.min and value <= clase.max :
                    diam=math.sqrt((clase.getMed()/escala)/math.pi)
                    break
            size=(spu.width()*diam)

    pathDistancia = QPainterPath()
    pathDistancia.arcTo(posX-(dist/2),posY-(dist/2),dist,dist,anguloInicio,tamAngulo/2)
    posPart = pathDistancia.currentPosition();
    x = posPart.x()
    y = posPart.y()
    if cont == 0:
        sumCalculated=False
        if n.atributosSectores:
            if not sumCalculated:
                suma=n.getSumSectores(feature)
                sumCalculated=True
            angSig=anguloInicio
            for camp in n.atributosSectores :
                simbolo=c.painter()
                simbolo.setBrush(camp.color)
                simbolo.setPen(self.mPen)
                simbolo.setOpacity(self.mSettingsDiagram.opacity)
                path = QPainterPath()
                angulo=((feature[camp.nombre]/suma)*tamAngulo)
                path.moveTo(x,y)
                path.arcTo(x-(size/2),y-(size/2),size,size,angSig,angulo)
                path.lineTo(x,y)
                simbolo.drawPath(path)
                angSig=angSig+angulo

            pathDistancia.arcTo(x-(size/2),y-(size/2),size,size,anguloInicio,tamAngulo/2)
            posText = pathDistancia.currentPosition();

            anguloText = anguloInicio+(tamAngulo/2)
            textHeight = 0
            textWidth = 0
            if(anguloText>90 and anguloText<270 ):
                textWidth = metrics.horizontalAdvance(n.campo.name())
            else :
                if(anguloText>=180):
                    textHeight = 10
            font = QFont('Arial', 6)
            simbolo.setFont(font)
            simbolo.drawText(posText.x()-textWidth,posText.y()+textHeight,n.campo.name())
            anguloInicio=angSig
        else:

            simbolo=c.painter()
            simbolo.setBrush(n.color)
            simbolo.setPen(self.mPen)
            simbolo.setOpacity(self.mSettingsDiagram.opacity)
            path = QPainterPath()
            pathDistancia = QPainterPath()
            path.moveTo(x,y)
            path.arcTo(x-(size/2),y-(size/2),size,size,anguloInicio,tamAngulo)
            if len(self.mSettingsDiagram.segmentosDeDiagrama)>1 :
                path.lineTo(x,y)
            simbolo.drawPath(path)
            pathDistancia.arcTo(x-(size/2),y-(size/2),size,size,anguloInicio,tamAngulo/2)
            posText = pathDistancia.currentPosition();

            anguloText = anguloInicio+(tamAngulo/2)

            anguloInicio=anguloInicio+tamAngulo

            textHeight = 0
            textWidth = 0
            if(anguloText>90 and anguloText<270 ):
                textWidth = metrics.horizontalAdvance(n.campo.name())
            else :
                if(anguloText>=180):
                    textHeight = 10
            font = QFont('Arial', 6)
            simbolo.setFont(font)
            simbolo.drawText(posText.x()-textWidth,posText.y()+textHeight,n.campo.name())
size=0
for n in self.mSettingsDiagram.segmentosCentrales:
    maxim=self.mSettingsDiagram.maxValSegmentosCentrales()
    minim=self.mSettingsDiagram.minValSegmentosCentrales()
    value=int(feature[n.campo.name()])
    if self.mSettingsDiagram.escalaContinuaT1:
        ajusTam=(maximunSizeT1/math.sqrt(maxim))
```

```
            size=(((math.sqrt(value)*ajusTam))*tam)
        else:
            clases=self.mSettingsDiagram.getClasesSegmentosCentrales()
            if clases:
                escala=1000*fixedSizeT1
                for clase in clases:
                    if value >= clase.min and value <= clase.max :
                        diam=math.sqrt((clase.getMed()/escala)/math.pi)

                size=diam

        simbolo=c.painter()
        simbolo.setBrush(n.color)
        simbolo.setOpacity(self.mSettingsDiagram.opacity)
        simbolo.drawEllipse(pos,size,size)
```

1.1.5 Clase atributosSectores

```
class atributosSectores:
    def __init__(self,nombre,color,etiqueta):
        self.nombre=nombre
        self.color=color
        self.etiqueta=etiqueta

    def getMin(self):
        return int(min(self.datos))

    def getMax(self):
        return int(max(self.datos))

    def getSum(self):
        return int(sum(self.datos))
```

1.1.6 Clase configuracionDiagrama

```
from PyQt5.QtGui import QColor
class configuracionDiagrama:
    def __init__(self):
        self.clasesSegmentosCentrales=[]
        self.clasesSegmentos=[]
        self.segmentosCentrales=[]
        self.segmentosDeDiagrama=[]
        self.maxValSC=0
        self.minValSC=0
        self.escalaContinuaT1=False
        self.escalaContinuaT2=False
        self.penColor=QColor()
        self.opacity=0.80
        self.penWidth=1
        self.fixedSizeT1=0
        self.fixedSizeT2=0
        self.maximunSizeT1=0
        self.maximunSizeT2=0
        self.distancia=0


    def setClasesSegmentosCentrales(self,clases):
        self.clasesSegmentosCentrales=clases

    def getClasesSegmentosCentrales(self):
        return self.clasesSegmentosCentrales

    def setClasesSegmentos(self,clases):
        self.clasesSegmentos=clases

    def getClasesSegmentos(self):
        return self.clasesSegmentos

    def maxValSegmentosDeDiagrama(self):
        maxim=0
        for n in self.segmentosDeDiagrama:
            if n.getMax()>maxim:
                maxim=n.getMax()
        return maxim

    def minValSegmentosDeDiagrama(self):
        minim=self.maxValSegmentosDeDiagrama()
        for n in self.segmentosDeDiagrama:
            if n.getMin()<minim:
                minim=n.getMin()
        return minim

    def maxValSegmentosCentrales(self):
```

```
        maxim=0
        for n in self.segmentosCentrales:
            if n.getMax()>maxim:
                maxim=n.getMax()
        return maxim

    def minValSegmentosCentrales(self):
        minim=self.maxValSegmentosCentrales()
        for n in self.segmentosCentrales:
            if n.getMin()<minim:
                minim=n.getMin()
        return minim
```

1.1.7 Clase EjesDiagrama

```
import Clase_Partes_Diagrama
class EjesDiagrama(Clase_Partes_Diagrama.PartesDiagrama):
        def __init__(self,idParte,color,etiqueta,campo,datos,clases,escCont):
            super().__init__(idParte,color,etiqueta,campo,datos)
            self.clases=clases
            self.escCont=escCont

        def getMin(self):
            return int(min(self.datos))

        def getMax(self):
            return int(max(self.datos))

        def getSum(self):
            return int(sum(self.datos))

        def getClases(self):
                    return self.clases

        def getEscalaContinua(self):
                    return self.escCont
```

1.1.8 Clase MetodosCalcClases

```
import math
from Clase_Renderer_Clase import *
class MetodosCalcClases:
    def __init__(self):
        self.datosAtributos=[]
        self.valTD=1
        self.datoMenor=0
        self.datoMayor=0
        self.totalDatos=0
        self.clases=[]

    def setDatosAtributos(self,datosAtributos):
        self.datosAtributos=datosAtributos
        self.datoMenor=int(min(datosAtributos))
        self.datoMayor=int(max(datosAtributos))
        self.totalDatos=len(datosAtributos)

    def metProgArit(self):
        self.clases=[]
        if self.datosAtributos:
            valorInicial=self.datoMenor
            valorFinal=valorInicial

            if self.datoMenor!=0:
                rProg=self.datoMenor
            else:
                self.datosAtributos.sort()
                pos=1
                while(self.datosAtributos[pos]==0 and pos<self.totalDatos):
                    pos=pos+1

                if  self.datosAtributos[pos]!=0:
                    rProg=self.datosAtributos[pos]
                else:
                    rProg=1

            while(valorInicial<self.datoMayor):
                valorFinal=valorFinal+rProg

                if valorFinal>self.datoMayor:
                    valorFinal=self.datoMayor

                nuevaClase=rendererClase(valorInicial,valorFinal)
                self.clases.append(nuevaClase)
                valorInicial=valorFinal+self.valTD
```

```
    return self.clases


def metProgGeom(self):
    self.clases=[]
    if self.datosAtributos:
        valorInicial=self.datoMenor
        valorFinal=valorInicial

        if self.datoMenor!=0:
            rProg=self.datoMenor
        else:
            valorFinal=1
            self.datosAtributos.sort()
            pos=1

            while(self.datosAtributos[pos]==0 and pos<self.totalDatos):
                pos=pos+1

            if  self.datosAtributos[pos]!=0 :
                rProg=self.datosAtributos[pos]
            else:
                rProg=1

        while(valorInicial<self.datoMayor):
            valorFinal=valorFinal*rProg

            if valorFinal>self.datoMayor:
                valorFinal=self.datoMayor

            nuevaClase=rendererClase(valorInicial,valorFinal)
            self.clases.append(nuevaClase)
            valorInicial=valorFinal+self.valTD

    return self.clases


def metProgGeomII(self,nClases):
    self.clases=[]
    if self.datosAtributos:
        valorInicial=self.datoMenor
        valorFinal=valorInicial

        if self.datoMenor!=0:
            rProg=math.pow((self.datoMayor/self.datoMenor),(1/nClases))
        else:
            rProg=math.pow(self.datoMayor,(1/nClases))
            valorFinal=1

        while(valorInicial<self.datoMayor):
            valorFinal=valorFinal*rProg

            if valorFinal>self.datoMayor:
                valorFinal=self.datoMayor

            nuevaClase=rendererClase(valorInicial,valorFinal)
            self.clases.append(nuevaClase)
            valorInicial=valorFinal+self.valTD

    return self.clases

def metClasesHomogeneas(self,nClases):
    self.clases=[]
    if self.datosAtributos:
        rProg=((self.datoMayor-self.datoMenor)/nClases)
        valorInicial=self.datoMenor
        valorFinal=valorInicial
        while(valorInicial<self.datoMayor):
            valorFinal=valorFinal+rProg

            if valorFinal>self.datoMayor:
                valorFinal=self.datoMayor

            nuevaClase=rendererClase(valorInicial,valorFinal)
            self.clases.append(nuevaClase)
            valorInicial=valorFinal+self.valTD

    return self.clases


def metClasesIrregulares(self):
    self.clases=[]
    if self.datosAtributos:
        nClases=math.floor(5*math.log10(self.totalDatos))
        nDatos=math.floor(self.totalDatos/nClases)
        datOrd=self.datosAtributos
        datOrd.sort(reverse=True)
        cont=0
        valorFinal=datOrd[cont]
```

```
            while(cont<self.totalDatos):
                cont+=nDatos
                if cont<self.totalDatos:
                    valorInicial=datOrd[cont]
                else:
                    valorInicial=datOrd[self.totalDatos-1]
                nuevaClase=rendererClase(valorInicial,valorFinal)
                self.clases.append(nuevaClase)

                if cont<self.totalDatos:
                    valorFinal=datOrd[cont+1]

        return self.clases


    def metLimitesDeClases(self):
        self.clases=[]
        if self.datosAtributos:
            nClases=math.floor(5*math.log10(len(self.datosAtributos)))

            if self.datoMenor!=0:
                kClases=((math.log10(self.datoMayor)-math.log10(self.datoMenor))/nClases)
            else:
                kClases=((math.log10(self.datoMayor))/nClases)

            datoTemp=self.datoMenor
            valorInicial=self.datoMenor
            n=0
            while n<nClases :
                if datoTemp!=0:
                    datoTemp=math.pow(10,math.log10(datoTemp)+kClases)
                else:
                    datoTemp=math.pow(10,kClases)

                if n+1==nClases:
                    nuevaClase=rendererClase(valorInicial,self.datoMayor)
                else:
                    nuevaClase=rendererClase(valorInicial,datoTemp)

                self.clases.append(nuevaClase)
                valorInicial=datoTemp+self.valTD
                n=n+1

        return self.clases
```

1.1.9 Clase SegmentosDiagrama

```
import Clase_Partes_Diagrama
class SegmentosDiagrama(Clase_Partes_Diagrama.PartesDiagrama):
    def __init__(self,idParte,color,etiqueta,campo,datos,atributosSectores):
        super().__init__(idParte,color,etiqueta,campo,datos)
        self.atributosSectores=atributosSectores


    def getMin(self):
        return int(min(self.datos))

    def getMax(self):
        return int(max(self.datos))

    def getSum(self):
        return int(sum(self.datos))

    def getSumSectores(self,feature):
        suma=0
        for atributos in self.atributosSectores:
            suma=suma+feature[atributos.nombre]
        return int(suma)
```

1.1.10 Clase rendererClase

```
class rendererClase:
    def __init__(self,valmin,valmax):
        self.min=int(valmin)
        self.max=int(valmax)

    def setMin(self,val):
        self.min=int(val)

    def setMax(self,val):
        self.max=int(val)

    def getMin(self):
        return self.min
```

```
    def getMax(self):
        return self.max

    def getMed(self):
        pMed=((self.min+self.max)/2)
        return pMed
```

1.1.11 Clase PartesDiagrama

```
class PartesDiagrama:
        def __init__(self,idParte,color,etiqueta,campo,datos):
            self.campo=campo
            self.idParte=idParte
            self.color=color
            self.etiqueta=etiqueta
            self.datos=datos

        def getMin(self):
            return int(min(self.datos))

        def getMax(self):
            return int(max(self.datos))

        def getSum(self):
            return int(sum(self.datos))
```

1.1.12 Clase agregarClases

```
import Clase_Clase_Box

from PyQt5.QtWidgets import QTableWidgetItem

def agregarClases(self,tablaClases,clases):
    for clase in clases:
        nuevaClase=Clase_Clase_Box.claseBox(clase)
        tablaClases.setRowCount(tablaClases.rowCount()+1)
        inter=str(clase.min)+"-"+str(clase.max)
        nuevoItemNo=QTableWidgetItem()
        nuevoItemNo.setText(str(tablaClases.rowCount()))
        tablaClases.setItem(tablaClases.rowCount()-1,0,nuevoItemNo)
        tablaClases.setCellWidget(tablaClases.rowCount()-1,1,nuevaClase)
```

1.1.13 Operaciones atributos

```
def geh(self):
     return int(min(self.datos))

def getMax(self):
    return int(max(self.datos))

def getSum(self):
    return int(sum(self.datos))


def cargarDatosAtributos(atributos,features):
    datosAtributos=[]
    for f in features:
        for atributo in atributos:
            datosAtributos.append(f[atributo.name()])
    return datosAtributos

def checkAtributoExistencia(atributo,listaAtributos):
    for atrib in listaAtributos:
        if atrib.name()==atributo:
            return atrib
    return None

def validarAtributo(atributo,features):
        for feat in features:
            dato=feat[atributo]
            try:
                value=int(dato)
            except ValueError:
                return False
        return True
```

1.1.14 Clase VentanaAgregarAtributosSectores

```
import random
from importlib import reload
from PyQt5.QtWidgets import QMainWindow,QTableWidgetItem,QMessageBox,QDialog,QAbstractItemView
import VentanaAgregarAtributosSectores
from Clase_Color_Box import *
from Clase_Atributos_Sectores import *
from qgis.utils import iface
from Clase_Operaciones_Atributos import *

reload(VentanaAgregarAtributosSectores)

class VentanaAgregarAtributosSectores(QDialog,VentanaAgregarAtributosSectores.Ui_Dialog):
    def __init__(self,layer,atributo,parent=None):
        super().__init__(parent)
        self.setupUi(self)
        self.setWindowTitle("Agregar sectores ( "+atributo+" )")
        self.atributos=[]
        self.layer=layer
        self.button_Add.clicked.connect(self.agregarAtributo)
        self.buttonDel.clicked.connect(self.quitarAtributo)
        self.buttonAceptar.clicked.connect(self.guardarAtributos)
        self.buttonCancelar.clicked.connect(self.cancelar)
        self.campos_capa.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.campos_capa.setDragEnabled(False)
        self.campos_sel.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.campos_sel.setDragEnabled(False)


    def agregarAtributo(self):
        if self.campos_capa.selectedItems() :
            color=[random.randint(180, 255), random.randint(180, 255), random.randint(180, 255)]
            nuevoColorBox=colorBox(defaultColor=QColor(color[0],color[1],color[2]))

            self.campos_sel.setRowCount(self.campos_sel.rowCount()+1)
            nombreAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])
            etiquetaAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,0,nombreAtributo)
            self.campos_sel.setCellWidget(self.campos_sel.rowCount()-1,1,nuevoColorBox)
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,2,etiquetaAtributo)


    def quitarAtributo(self):
        if self.campos_sel.selectedItems() :
            self.campos_sel.removeRow(self.campos_sel.currentRow())

    def guardarAtributos(self):
        self.atributos=[]
        for n in range(self.campos_sel.rowCount()) :
            nombre=self.campos_sel.item(n,0).text()
            atributoNuevo=atributosSectores(nombre,self.campos_sel.cellWidget(n,1).color(),self.campos_sel.item(n,2).text())
            self.atributos.append(atributoNuevo)
        self.close()

    def cancelar(self):
        self.campos_sel.setRowCount(1)
        self.campos_sel.removeRow(0)
        for atributo in self.atributos :
            self.campos_sel.setRowCount(self.campos_sel.rowCount()+1)
            nombreAtributo=QTableWidgetItem()
            nombreAtributo.setText(atributo.nombre)
            etiquetaAtributo=QTableWidgetItem()
            etiquetaAtributo.setText(atributo.etiqueta)
            nuevoColorBox=colorBox(defaultColor=atributo.color)
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,0,nombreAtributo)
            self.campos_sel.setCellWidget(self.campos_sel.rowCount()-1,1,nuevoColorBox)
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,2,etiquetaAtributo)
        self.close()
```

1.1.15 Clase VentanaAgregarClases

```
import random
import Clase_Ventana_Intervalo_De_Clases
import Clase_Clase_Box
import Clase_Operaciones_Tablas_Clases
import Clase_Operaciones_Atributos
import Clase_Metodos_Calc_Clases
import VentanaAgregarClases
from importlib import reload
reload(Clase_Operaciones_Atributos)
reload(Clase_Operaciones_Tablas_Clases)
reload(Clase_Metodos_Calc_Clases)
reload(VentanaAgregarClases)
#import sys
from PyQt5.QtWidgets import QDialog,QTableWidgetItem,QAbstractItemView

class VentanaAgregarClases(QDialog,VentanaAgregarClases.Ui_Dialog):
    def __init__(self,atributo,parent=None):
        super().__init__(parent)
```

```
    self.setupUi(self)
    self.setWindowTitle("Clases ( " + atributo +" )")
    self.atributos=[]
    self.features=[]
    self.datos=[]
    self.clases=[]
    self.metodoClases=Clase_Metodos_Calc_Clases.MetodosCalcClases()
    self.textNClases.setEnabled(False)
    self.textNClases.setValue(1)
    self.btnDelAllClase.clicked.connect(self.deleteAllClases)
    self.btnAddClase.clicked.connect(self.agregarClase)
    self.btnDelClase.clicked.connect(self.deleteClase)
    self.aceptarButton.clicked.connect(self.guardarClases)
    self.cancelarButton.clicked.connect(self.cancelar)
    self.comboBox.activated.connect(self.actualizarClases)
    self.textNClases.valueChanged.connect(self.actualizarClases)
    self.tableClases.setEditTriggers(QAbstractItemView.NoEditTriggers)
    self.tableClases.setDragEnabled(False)


def actualizarClases(self):
    idMetodo=self.comboBox.currentIndex()
    if idMetodo==3 or idMetodo==4:
        self.textNClases.setEnabled(True)
    else:
        self.textNClases.setEnabled(False)


    self.agregarClasesTabla(self.tableClases,self.crearClases())



def deleteClase(self):
    if self.tableClases.selectedItems() :
        self.tableClases.removeRow(self.tableClases.currentRow())

def deleteAllClases(self):
    self.tableClases.setRowCount(1)
    self.tableClases.removeRow(0)

def agregarClase(self):
    clase=Clase_Ventana_Intervalo_De_Clases.VentanaIntervaloClases.setClase()
    if clase:
        nuevaClase=Clase_Clase_Box.claseBox(clase)
        self.tableClases.setRowCount(self.tableClases.rowCount()+1)
        nuevoItemNo=QTableWidgetItem()
        nuevoItemNo.setText(str(self.tableClases.rowCount()))
        self.tableClases.setItem(self.tableClases.rowCount()-1,0,nuevoItemNo)
        self.tableClases.setCellWidget(self.tableClases.rowCount()-1,1,nuevaClase)

def agregarClasesTabla(self,tablaClases,clases):
    self.deleteAllClases()
    for clase in clases:
        nuevaClase=Clase_Clase_Box.claseBox(clase)
        tablaClases.setRowCount(tablaClases.rowCount()+1)
        inter=str(clase.min)+"-"+str(clase.max)
        nuevoItemNo=QTableWidgetItem()
        nuevoItemNo.setText(str(tablaClases.rowCount()))
        tablaClases.setItem(tablaClases.rowCount()-1,0,nuevoItemNo)
        tablaClases.setCellWidget(tablaClases.rowCount()-1,1,nuevaClase)

def crearClases(self):
    idMetodo=self.comboBox.currentIndex()
    cant=int(self.textNClases.text())
    if cant==0:
        cant=1

    if idMetodo==0:
        clases=self.metodoClases.metLimitesDeClases()
    elif idMetodo==1:
        clases=self.metodoClases.metProgArit()
    elif idMetodo==2:
        clases=self.metodoClases.metProgGeom()
    elif idMetodo==3:
        clases=self.metodoClases.metProgGeomII(cant)
    elif idMetodo==4:
        clases=self.metodoClases.metClasesHomogeneas(cant)
    elif idMetodo==5:
        clases=self.metodoClases.metClasesIrregulares()

    return clases


def insertarDatos(self,atributos,features):
    self.atributos=atributos
    self.features=features
    self.datos=Clase_Operaciones_Atributos.cargarDatosAtributos(atributos,features)
    self.metodoClases.setDatosAtributos(self.datos)
    self.clases=self.crearClases()
    self.agregarClasesTabla(self.tableClases,self.clases)
```

```
    def getClases(self,tabAttrib):
        clases=[]
        for n in range(tabAttrib.rowCount()) :
            clase=tabAttrib.cellWidget(n,1).getClase()
            clases.append(clase)
        return clases


    def guardarClases(self):
        self.clases=self.getClases(self.tableClases)
        self.close()

    def cancelar(self):
        self.close()
```

1.1.16 Clase VentanaClases

```
from importlib import reload
import Clase_Renderer_Clase
from VentanaClases import *
from PyQt5.QtGui import QDoubleValidator
reload(Clase_Renderer_Clase)
class VentanaClases(QtWidgets.QDialog,Ui_Dialog):
    def __init__(self,parent=None):
        super(VentanaClases,self).__init__(parent)
        self.setupUi(self)
        self.setWindowTitle("Modificar Valores de Clase")
        self.onlyInt = QDoubleValidator()
        self.valorInicial.setValidator(self.onlyInt)
        self.ValorFinal.setValidator(self.onlyInt)

        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)

    @staticmethod
    def getClase(clase):
        VC=VentanaClases()
        result=VC.exec()
        if result==1 and VC.valorInicial.text() and VC.ValorFinal.text():
            n= VC.modificarClase(clase,VC.valorInicial.text(),VC.ValorFinal.text())
            return n
        else:
            return clase

    @staticmethod
    def setClase():

        VC=VentanaClases()
        result=VC.exec()
        if result==1 and VC.valorInicial.text() and VC.ValorFinal.text():
            return VC.crearClase(VC.valorInicial.text(),VC.ValorFinal.text())
        else:
            return None

    def modificarClase(self,clase,minVal,maxVal):
        clase.setMin(minVal)
        clase.setMax(maxVal)
        return clase

    def crearClase(self,minVal,maxVal):
        return Clase_Renderer_Clase.rendererClase(minVal,maxVal)
```

1.1.17 Clase VentanaIntervaloClases

```
from importlib import reload
import Clase_Renderer_Clase
from VentanaIntervaloClases import *
from PyQt5.QtGui import QDoubleValidator
reload(Clase_Renderer_Clase)
class VentanaIntervaloClases(QtWidgets.QDialog,Ui_Dialog):
    def __init__(self,parent=None):
        super(VentanaIntervaloClases,self).__init__(parent)
        self.setupUi(self)
        self.setWindowTitle("Modificar Valores de Clase")
        self.onlyInt = QDoubleValidator()
        self.valorInicial.setValidator(self.onlyInt)
        self.ValorFinal.setValidator(self.onlyInt)

        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)
```

```
    @staticmethod
    def getClase(clase):
        VC=VentanaIntervaloClases()
        result=VC.exec()
        if result==1 and VC.valorInicial.text() and VC.ValorFinal.text():
            return VC.modificarClase(clase,VC.valorInicial.text(),VC.ValorFinal.text())
        else:
            return clase

    @staticmethod
    def setClase():
        VC=VentanaIntervaloClases()
        result=VC.exec()
        if result==1 and VC.valorInicial.text() and VC.ValorFinal.text():
            return VC.crearClase(VC.valorInicial.text(),VC.ValorFinal.text())
        else:
            return None

    def modificarClase(self,clase,minVal,maxVal):
        clase.setMin(minVal)
        clase.setMax(maxVal)
        return clase

    def crearClase(self,minVal,maxVal):
        return Clase_Renderer_Clase.rendererClase(minVal,maxVal)
```

1.1.18 Clase VentanaAgregarAtributosSectores

```
import random
from PyQt5.QtWidgets import QMainWindow,QTableWidgetItem
from VentanaAgregarAtributosSectores import *
from Clase_Color_Box import *
from Clase_Atributos_Sectores import *

class VentanaAgregarAtributosSectores(QMainWindow,Ui_MainWindow):
    def __init__(self,*args,**kwargs):
        QtWidgets.QMainWindow.__init__(self,*args,**kwargs)
        self.setupUi(self)
        self.setWindowTitle("Agregar atributos")
        self.atributos=[]
        self.button_Add.clicked.connect(self.agregarAtributo)
        self.buttonDel.clicked.connect(self.quitarAtributo)
        self.buttonAceptar.clicked.connect(self.guardarAtributos)


    def agregarAtributo(self):
        if self.campos_capa.selectedItems() :
            color=[random.randint(180, 255), random.randint(180, 255), random.randint(180, 255)]
            colorBox=ColorBox(defaultColor=QColor(color[0],color[1],color[2]))

            self.campos_sel.setRowCount(self.campos_sel.rowCount()+1)
            nombreAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])
            etiquetaAtributo=QTableWidgetItem(self.campos_capa.selectedItems()[0])
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,0,nombreAtributo)
            self.campos_sel.setCellWidget(self.campos_sel.rowCount()-1,1,colorBox)
            self.campos_sel.setItem(self.campos_sel.rowCount()-1,2,etiquetaAtributo)

    def quitarAtributo(self):
        if self.campos_sel.selectedItems() :
            self.campos_sel.removeRow(self.campos_sel.currentRow())

    def guardarAtributos(self):
        self.atributos=[]
        for n in range(self.campos_sel.rowCount()) :
            nombre=self.campos_sel.item(n,0).text()
            atributoNuevo=AtributosSectores(nombre,self.campos_sel.cellWidget(n,1).color(),self.campos_sel.item(n,2).text())
            self.atributos.append(atributoNuevo)
```

1.1.19 Clase colorBox

```
from PyQt5.QtWidgets import QFrame, QColorDialog
from PyQt5.QtCore import Qt, pyqtSignal
from PyQt5.QtGui import QColor

class colorBox(QFrame):

    colorChanged = pyqtSignal()

    def __init__(self, parent=None, defaultColor=Qt.white):
        super(colorBox,self).__init__(parent)
        self.setStyleSheet("border-color: rgba(0,0,0,0);")
        self.__color = None
        self.__defaultColor = QColor(defaultColor)
        self.setColor(self.__defaultColor)
```

```
        self.setFixedHeight(20)
        self.setFrameStyle(1)

    def setColor(self, color):
        if color != self.__color:
            self.__color = QColor(color)
            self.setStyleSheet("background-color: %s;" % self.__color.name())
            self.colorChanged.emit()

    def color(self):
        return self.__color

    def mousePressEvent(self, e):
        if e.buttons() == Qt.LeftButton:
            color = QColorDialog.getColor(self.__color)
            if color.isValid():
                self.setColor(color)
                self.colorChanged.emit()
        elif e.button() == Qt.RightButton:
            self.setColor(self.__defaultColor)
```

1.1.20 Clase claseBox

```
import Clase_Ventana_Intervalo_De_Clases
from Clase_Renderer_Clase import *
from PyQt5.QtWidgets import QLineEdit
from PyQt5.QtCore import Qt, pyqtSignal
from PyQt5.QtGui import QColor
from importlib import reload
reload(Clase_Ventana_Intervalo_De_Clases)
class claseBox(QLineEdit):

    rangeChanged = pyqtSignal()

    def __init__(self,clase):
        super(claseBox,self).__init__()
        self.clase=clase
        self.setFrame(False)
        self.setText(str(clase.min)+" - "+str(clase.max))
        self.setReadOnly(True)


    def setClase(self, clase):
        self.clase= clase
        self.setText(str(clase.min)+" - "+str(clase.max))
        self.rangeChanged.emit()

    def getClase(self):
        return self.clase

    def mouseDoubleClickEvent(self, e):
        if e.buttons() == Qt.LeftButton:
            claseTemp = Clase_Ventana_Intervalo_De_Clases.VentanaIntervaloClases.getClase(self.clase)
            self.setClase(claseTemp)
            self.rangeChanged.emit()

        elif e.button() == Qt.RightButton:
            pass
```

1.1.21 Clase BtnAgregarAtributosSectores

```
from PyQt5.QtWidgets import QPushButton, QTableWidgetItem
from PyQt5.QtCore import Qt, pyqtSignal
from PyQt5.QtGui import QColor
from importlib import reload
import Clase_Ventana_Agregar_Atributos_Sectores

reload(Clase_Ventana_Agregar_Atributos_Sectores)

class BtnAgregarAtributosSectores(QPushButton):

    agregarAtributos = pyqtSignal()

    def __init__(self,atributo,tablaCampos,layer,parent=None):
        super().__init__(parent)
        self.setText("Agregar")
        self.parent=parent
        self.ventanaSectores=Clase_Ventana_Agregar_Atributos_Sectores.VentanaAgregarAtributosSectores(layer,atributo)

        for n in range(tablaCampos.rowCount()):
            nombreAtributo=QTableWidgetItem(tablaCampos.item(n,0))
            self.ventanaSectores.campos_capa.setRowCount(tablaCampos.rowCount())
            self.ventanaSectores.campos_capa.setItem(n,0,nombreAtributo)
```

```
    def mousePressEvent(self, e):
        if e.buttons() == Qt.LeftButton:
            self.ventanaSectores.exec()
            self.agregarAtributos.emit()
```

1.1.22 Clase BtnAgregarClases

```
from PyQt5.QtWidgets import QPushButton, QTableWidgetItem
from PyQt5.QtCore import Qt, pyqtSignal
from PyQt5.QtGui import QColor
from importlib import reload
import Clase_Ventana_Agregar_Clases
import Clase_Operaciones_Tablas_Clases
import Clase_Metodos_Calc_Clases
reload(Clase_Ventana_Agregar_Clases)
reload(Clase_Metodos_Calc_Clases)
reload(Clase_Ventana_Agregar_Clases)

class BtnAgregarClases(QPushButton):

    agregarClases = pyqtSignal()

    def __init__(self,atributo,parent=None):
        super().__init__(parent)
        self.setText("Ver clases")
        self.ventanaClases=Clase_Ventana_Agregar_Clases.VentanaAgregarClases(atributo)


    def mousePressEvent(self, e):
        if e.buttons() == Qt.LeftButton:
                self.ventanaClases.agregarClasesTabla(self.ventanaClases.tableClases,self.ventanaClases.clases)
                self.ventanaClases.datoMayor.setText(str(self.ventanaClases.metodoClases.datoMayor))
                self.ventanaClases.label_4.setText(str(self.ventanaClases.metodoClases.datoMenor))
                        self.ventanaClases.exec()
                self.agregarClases.emit()
```

1.2 Interfaces de usuario

1.2.1 VentanaAgregarAtributosSectores

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.setFixedSize(571, 370)
        Dialog.setSizeGripEnabled(False)
        Dialog.setModal(False)
        self.buttonAceptar = QtWidgets.QPushButton(Dialog)
        self.buttonAceptar.setGeometry(QtCore.QRect(380, 330, 75, 23))
        self.buttonAceptar.setObjectName("buttonAceptar")
        self.groupBox = QtWidgets.QGroupBox(Dialog)
        self.groupBox.setGeometry(QtCore.QRect(30, 20, 511, 291))
        self.groupBox.setObjectName("groupBox")
        self.campos_capa = QtWidgets.QTableWidget(self.groupBox)
        self.campos_capa.setEnabled(True)
        self.campos_capa.setGeometry(QtCore.QRect(10, 50, 141, 221))
        self.campos_capa.setMouseTracking(True)
        self.campos_capa.setTabletTracking(True)
        self.campos_capa.setFocusPolicy(QtCore.Qt.NoFocus)
        self.campos_capa.setAcceptDrops(True)
        self.campos_capa.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.campos_capa.setAutoFillBackground(True)
        self.campos_capa.setFrameShape(QtWidgets.QFrame.Box)
        self.campos_capa.setLineWidth(1)
        self.campos_capa.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents)
        self.campos_capa.setAutoScroll(True)
        self.campos_capa.setDragEnabled(True)
        self.campos_capa.setAlternatingRowColors(False)
        self.campos_capa.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
        self.campos_capa.setShowGrid(False)
        self.campos_capa.setGridStyle(QtCore.Qt.SolidLine)
        self.campos_capa.setWordWrap(True)
        self.campos_capa.setCornerButtonEnabled(False)
        self.campos_capa.setColumnCount(1)
        self.campos_capa.setProperty("centrarTexto", True)
        self.campos_capa.setObjectName("campos_capa")
        self.campos_capa.setRowCount(0)
```

```
item = QtWidgets.QTableWidgetItem()
self.campos_capa.setHorizontalHeaderItem(0, item)
self.campos_capa.horizontalHeader().setVisible(True)
self.campos_capa.horizontalHeader().setCascadingSectionResizes(False)
self.campos_capa.horizontalHeader().setDefaultSectionSize(120)
self.campos_capa.horizontalHeader().setMinimumSectionSize(20)
self.campos_capa.horizontalHeader().setSortIndicatorShown(True)
self.campos_capa.horizontalHeader().setStretchLastSection(True)
self.campos_capa.verticalHeader().setVisible(False)
self.campos_capa.verticalHeader().setCascadingSectionResizes(False)
self.campos_capa.verticalHeader().setDefaultSectionSize(15)
self.campos_capa.verticalHeader().setHighlightSections(True)
self.campos_capa.verticalHeader().setSortIndicatorShown(True)
self.button_Add = QtWidgets.QPushButton(self.groupBox)
self.button_Add.setGeometry(QtCore.QRect(160, 50, 31, 31))
self.button_Add.setObjectName("button_Add")
self.buttonDel = QtWidgets.QPushButton(self.groupBox)
self.buttonDel.setGeometry(QtCore.QRect(160, 90, 31, 31))
self.buttonDel.setObjectName("buttonDel")
self.campos_sel = QtWidgets.QTableWidget(self.groupBox)
self.campos_sel.setEnabled(True)
self.campos_sel.setGeometry(QtCore.QRect(200, 50, 291, 221))
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
sizePolicy.setHorizontalStretch(100)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.campos_sel.sizePolicy().hasHeightForWidth())
self.campos_sel.setSizePolicy(sizePolicy)
font = QtGui.QFont()
font.setBold(False)
font.setWeight(50)
font.setKerning(True)
font.setStyleStrategy(QtGui.QFont.PreferDefault)
self.campos_sel.setFont(font)
self.campos_sel.setMouseTracking(True)
self.campos_sel.setTabletTracking(True)
self.campos_sel.setFocusPolicy(QtCore.Qt.NoFocus)
self.campos_sel.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)
self.campos_sel.setAcceptDrops(True)
self.campos_sel.setToolTip("")
self.campos_sel.setLayoutDirection(QtCore.Qt.LeftToRight)
self.campos_sel.setAutoFillBackground(True)
self.campos_sel.setFrameShape(QtWidgets.QFrame.Box)
self.campos_sel.setLineWidth(1)
self.campos_sel.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents)
self.campos_sel.setAutoScroll(True)
self.campos_sel.setAutoScrollMargin(16)
self.campos_sel.setTabKeyNavigation(True)
self.campos_sel.setProperty("showDropIndicator", True)
self.campos_sel.setDragEnabled(True)
self.campos_sel.setAlternatingRowColors(False)
self.campos_sel.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
self.campos_sel.setTextElideMode(QtCore.Qt.ElideMiddle)
self.campos_sel.setShowGrid(False)
self.campos_sel.setGridStyle(QtCore.Qt.NoPen)
self.campos_sel.setWordWrap(True)
self.campos_sel.setCornerButtonEnabled(False)
self.campos_sel.setColumnCount(3)
self.campos_sel.setProperty("centrarTexto", True)
self.campos_sel.setObjectName("campos_sel")
self.campos_sel.setRowCount(0)
item = QtWidgets.QTableWidgetItem()
self.campos_sel.setHorizontalHeaderItem(0, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel.setHorizontalHeaderItem(1, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel.setHorizontalHeaderItem(2, item)
self.campos_sel.horizontalHeader().setVisible(True)
self.campos_sel.horizontalHeader().setCascadingSectionResizes(False)
self.campos_sel.horizontalHeader().setDefaultSectionSize(95)
self.campos_sel.horizontalHeader().setMinimumSectionSize(20)
self.campos_sel.horizontalHeader().setSortIndicatorShown(True)
self.campos_sel.horizontalHeader().setStretchLastSection(True)
self.campos_sel.verticalHeader().setVisible(False)
self.campos_sel.verticalHeader().setCascadingSectionResizes(False)
self.campos_sel.verticalHeader().setDefaultSectionSize(15)
self.campos_sel.verticalHeader().setHighlightSections(True)
self.campos_sel.verticalHeader().setMinimumSectionSize(10)
self.campos_sel.verticalHeader().setSortIndicatorShown(True)
self.label = QtWidgets.QLabel(self.groupBox)
self.label.setGeometry(QtCore.QRect(11, 30, 121, 16))
font = QtGui.QFont()
font.setPointSize(9)
self.label.setFont(font)
self.label.setObjectName("label")
self.label_2 = QtWidgets.QLabel(self.groupBox)
self.label_2.setGeometry(QtCore.QRect(202, 30, 161, 16))
font = QtGui.QFont()
font.setPointSize(9)
self.label_2.setFont(font)
self.label_2.setObjectName("label_2")
self.buttonCancelar = QtWidgets.QPushButton(Dialog)
```

```
        self.buttonCancelar.setGeometry(QtCore.QRect(470, 330, 75, 23))
        self.buttonCancelar.setObjectName("buttonCancelar")
        self.menubar = QtWidgets.QMenuBar(Dialog)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 571, 21))
        self.menubar.setObjectName("menubar")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.buttonAceptar.setText(_translate("Dialog", "Aceptar"))
        self.groupBox.setTitle(_translate("Dialog", "Attributos"))
        self.campos_capa.setSortingEnabled(True)
        item = self.campos_capa.horizontalHeaderItem(0)
        item.setText(_translate("Dialog", "Atributos"))
        self.button_Add.setText(_translate("Dialog", "Add"))
        self.buttonDel.setText(_translate("Dialog", "Del"))
        self.campos_sel.setSortingEnabled(True)
        item = self.campos_sel.horizontalHeaderItem(0)
        item.setText(_translate("Dialog", "Atributos"))
        item = self.campos_sel.horizontalHeaderItem(1)
        item.setText(_translate("Dialog", "Color"))
        item = self.campos_sel.horizontalHeaderItem(2)
        item.setText(_translate("Dialog", "Etiqueta"))
        self.label.setText(_translate("Dialog", "Atributos Disponibles"))
        self.label_2.setText(_translate("Dialog", "Atributo(s) Seleccionados"))
        self.buttonCancelar.setText(_translate("Dialog", "Cancelar"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

1.2.2 VentanaAgregarClases

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.setFixedSize(441, 450)
        Dialog.setSizeGripEnabled(False)
        Dialog.setModal(False)
        self.groupBox_2 = QtWidgets.QGroupBox(Dialog)
        self.groupBox_2.setGeometry(QtCore.QRect(20, 80, 401, 331))
        self.groupBox_2.setObjectName("groupBox_2")
        self.tableClases = QtWidgets.QTableWidget(self.groupBox_2)
        self.tableClases.setGeometry(QtCore.QRect(20, 25, 371, 211))
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.tableClases.sizePolicy().hasHeightForWidth())
        self.tableClases.setSizePolicy(sizePolicy)
        self.tableClases.setFrameShape(QtWidgets.QFrame.Box)
        self.tableClases.setSelectionMode(QtWidgets.QAbstractItemView.ExtendedSelection)
        self.tableClases.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
        self.tableClases.setObjectName("tableClases")
        self.tableClases.setColumnCount(2)
        self.tableClases.setRowCount(0)
        item = QtWidgets.QTableWidgetItem()
        self.tableClases.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.tableClases.setHorizontalHeaderItem(1, item)
        self.tableClases.horizontalHeader().setVisible(True)
        self.tableClases.horizontalHeader().setCascadingSectionResizes(False)
        self.tableClases.horizontalHeader().setDefaultSectionSize(70)
        self.tableClases.horizontalHeader().setHighlightSections(True)
        self.tableClases.horizontalHeader().setMinimumSectionSize(20)
        self.tableClases.horizontalHeader().setStretchLastSection(True)
        self.tableClases.verticalHeader().setVisible(False)
        self.tableClases.verticalHeader().setDefaultSectionSize(15)
        self.tableClases.verticalHeader().setMinimumSectionSize(10)
        self.btnAddClase = QtWidgets.QPushButton(self.groupBox_2)
        self.btnAddClase.setGeometry(QtCore.QRect(20, 270, 75, 23))
        self.btnAddClase.setObjectName("btnAddClase")
        self.btnDelClase = QtWidgets.QPushButton(self.groupBox_2)
        self.btnDelClase.setGeometry(QtCore.QRect(110, 270, 75, 23))
        self.btnDelClase.setObjectName("btnDelClase")
        self.btnDelAllClase = QtWidgets.QPushButton(self.groupBox_2)
        self.btnDelAllClase.setGeometry(QtCore.QRect(200, 270, 75, 23))
        self.btnDelAllClase.setObjectName("btnDelAllClase")
```

```
        self.checkEscContCC = QtWidgets.QCheckBox(self.groupBox_2)
        self.checkEscContCC.setGeometry(QtCore.QRect(20, 300, 200, 17))
        self.checkEscContCC.setObjectName("checkEscContCC")
        self.label = QtWidgets.QLabel(self.groupBox_2)
        self.label.setGeometry(QtCore.QRect(20, 240, 90, 16))
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.groupBox_2)
        self.label_2.setGeometry(QtCore.QRect(210, 240, 90, 16))
        self.label_2.setObjectName("label_2")
        self.datoMayor = QtWidgets.QLabel(self.groupBox_2)
        self.datoMayor.setGeometry(QtCore.QRect(105, 240, 90, 16))
        self.datoMayor.setText("")
        self.datoMayor.setObjectName("datoMayor")
        self.label_4 = QtWidgets.QLabel(self.groupBox_2)
        self.label_4.setGeometry(QtCore.QRect(295, 240, 90, 20))
        self.label_4.setText("")
        self.label_4.setObjectName("label_4")
        self.comboBox = QtWidgets.QComboBox(Dialog)
        self.comboBox.setGeometry(QtCore.QRect(200, 15, 211, 22))
        self.comboBox.setObjectName("comboBox")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.label_5 = QtWidgets.QLabel(Dialog)
        self.label_5.setGeometry(QtCore.QRect(30, 15, 181, 16))
        self.label_5.setObjectName("label_5")
        self.label_6 = QtWidgets.QLabel(Dialog)
        self.label_6.setGeometry(QtCore.QRect(30, 50, 120, 16))
        self.label_6.setObjectName("label_6")
        self.textNClases = QtWidgets.QSpinBox(Dialog)
        self.textNClases.setGeometry(QtCore.QRect(200, 50, 211, 22))
        self.textNClases.setObjectName("textNClases")
        self.cancelarButton = QtWidgets.QPushButton(Dialog)
        self.cancelarButton.setGeometry(QtCore.QRect(342, 420, 80, 23))
        self.cancelarButton.setObjectName("cancelarButton")
        self.aceptarButton = QtWidgets.QPushButton(Dialog)
        self.aceptarButton.setGeometry(QtCore.QRect(250, 420, 80, 23))
        self.aceptarButton.setObjectName("aceptarButton")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.groupBox_2.setTitle(_translate("Dialog", "Clases"))
        item = self.tableClases.horizontalHeaderItem(0)
        item.setText(_translate("Dialog", "No. Clase"))
        item = self.tableClases.horizontalHeaderItem(1)
        item.setText(_translate("Dialog", "Intervalo"))
        self.btnAddClase.setText(_translate("Dialog", "Add"))
        self.btnDelClase.setText(_translate("Dialog", "Del"))
        self.btnDelAllClase.setText(_translate("Dialog", "Delete All"))
        self.checkEscContCC.setText(_translate("Dialog", "Datos en escala continua"))
        self.label.setText(_translate("Dialog", "Dato mayor:"))
        self.label_2.setText(_translate("Dialog", "Dato Menor:"))
        self.comboBox.setItemText(0, _translate("Dialog", "Intervalos irregulares o variables"))
        self.comboBox.setItemText(1, _translate("Dialog", "Progresion aritmetica"))
        self.comboBox.setItemText(2, _translate("Dialog", "Progresion geometrica"))
        self.comboBox.setItemText(3, _translate("Dialog", "Progresion geometrica (Razon progresion)"))
        self.comboBox.setItemText(4, _translate("Dialog", "Distrubucion de frecuencias absolutas"))
        self.comboBox.setItemText(5, _translate("Dialog", "Clases irregulares (numero igual de datos)"))
        self.label_5.setText(_translate("Dialog", "Metodo de clases"))
        self.label_6.setText(_translate("Dialog", "Numero de clases"))
        self.cancelarButton.setText(_translate("Dialog", "Cancelar"))
        self.aceptarButton.setText(_translate("Dialog", "Aceptar"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

1.2.3 VentanaClases

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
```

```
        Dialog.setFixedSize(265, 169)
        self.label = QtWidgets.QLabel(Dialog)
        self.label.setGeometry(QtCore.QRect(20, 30, 81, 16))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(Dialog)
        self.label_2.setGeometry(QtCore.QRect(20, 80, 71, 16))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.valorInicial = QtWidgets.QLineEdit(Dialog)
        self.valorInicial.setGeometry(QtCore.QRect(110, 30, 131, 20))
        self.valorInicial.setObjectName("valorInicial")
        self.ValorFinal = QtWidgets.QLineEdit(Dialog)
        self.ValorFinal.setGeometry(QtCore.QRect(110, 80, 131, 20))
        self.ValorFinal.setObjectName("ValorFinal")
        self.buttonBox = QtWidgets.QDialogButtonBox(Dialog)
        self.buttonBox.setGeometry(QtCore.QRect(110, 130, 131, 23))
        self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok)
        self.buttonBox.setObjectName("buttonBox")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.label.setText(_translate("Dialog", "Valor inicial"))
        self.label_2.setText(_translate("Dialog", "Valor final"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

1.2.4 VentanaInicial

```
from PyQt5 import QtCore, QtGui, QtWidgets
import Clase_Btn_Agregar_Clases

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.setWindowModality(QtCore.Qt.NonModal)
        MainWindow.setFixedSize(619, 497)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.diagramSel = QtWidgets.QComboBox(self.centralwidget)
        self.diagramSel.setGeometry(QtCore.QRect(30, 10, 561, 22))
        self.diagramSel.setObjectName("diagramSel")
        self.diagramSel.addItem("")
        self.diagramSel.addItem("")
        self.diagramSel.addItem("")
        self.buttonAceptar = QtWidgets.QPushButton(self.centralwidget)
        self.buttonAceptar.setGeometry(QtCore.QRect(355, 470, 75, 23))
        self.buttonAceptar.setObjectName("buttonAceptar")
        self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
        self.tabWidget.setGeometry(QtCore.QRect(30, 40, 561, 421))
        self.tabWidget.setTabPosition(QtWidgets.QTabWidget.North)
        self.tabWidget.setTabShape(QtWidgets.QTabWidget.Rounded)
        self.tabWidget.setElideMode(QtCore.Qt.ElideNone)
        self.tabWidget.setObjectName("tabWidget")
        self.atributos = QtWidgets.QWidget()
        self.atributos.setObjectName("atributos")
        self.groupBox = QtWidgets.QGroupBox(self.atributos)
        self.groupBox.setGeometry(QtCore.QRect(20, 10, 511, 371))
        self.groupBox.setObjectName("groupBox")
        self.campos_capa = QtWidgets.QTableWidget(self.groupBox)
        self.campos_capa.setEnabled(True)
        self.campos_capa.setGeometry(QtCore.QRect(10, 50, 151, 301))
        self.campos_capa.setMouseTracking(True)
        self.campos_capa.setTabletTracking(True)
        self.campos_capa.setFocusPolicy(QtCore.Qt.NoFocus)
        self.campos_capa.setAcceptDrops(True)
        self.campos_capa.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.campos_capa.setAutoFillBackground(True)
        self.campos_capa.setFrameShape(QtWidgets.QFrame.Box)
        self.campos_capa.setLineWidth(1)
        self.campos_capa.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents)
        self.campos_capa.setAutoScroll(True)
```

```
self.campos_capa.setDragEnabled(True)
self.campos_capa.setAlternatingRowColors(False)
self.campos_capa.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
self.campos_capa.setShowGrid(False)
self.campos_capa.setGridStyle(QtCore.Qt.SolidLine)
self.campos_capa.setWordWrap(True)
self.campos_capa.setCornerButtonEnabled(False)
self.campos_capa.setColumnCount(1)
self.campos_capa.setProperty("centrarTexto", True)
self.campos_capa.setObjectName("campos_capa")
self.campos_capa.setRowCount(0)
item = QtWidgets.QTableWidgetItem()
self.campos_capa.setHorizontalHeaderItem(0, item)
self.campos_capa.horizontalHeader().setVisible(True)
self.campos_capa.horizontalHeader().setCascadingSectionResizes(False)
self.campos_capa.horizontalHeader().setDefaultSectionSize(100)
self.campos_capa.horizontalHeader().setMinimumSectionSize(20)
self.campos_capa.horizontalHeader().setSortIndicatorShown(True)
self.campos_capa.horizontalHeader().setStretchLastSection(True)
self.campos_capa.verticalHeader().setVisible(False)
self.campos_capa.verticalHeader().setCascadingSectionResizes(False)
self.campos_capa.verticalHeader().setDefaultSectionSize(15)
self.campos_capa.verticalHeader().setHighlightSections(True)
self.campos_capa.verticalHeader().setSortIndicatorShown(True)
self.button_Add = QtWidgets.QPushButton(self.groupBox)
self.button_Add.setGeometry(QtCore.QRect(170, 50, 31, 31))
self.button_Add.setObjectName("button_Add")
self.buttonDel = QtWidgets.QPushButton(self.groupBox)
self.buttonDel.setGeometry(QtCore.QRect(170, 90, 31, 31))
self.buttonDel.setObjectName("buttonDel")
self.campos_sel = QtWidgets.QTableWidget(self.groupBox)
self.campos_sel.setEnabled(True)
self.campos_sel.setGeometry(QtCore.QRect(210, 50, 291, 71))
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
sizePolicy.setHorizontalStretch(100)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.campos_sel.sizePolicy().hasHeightForWidth())
self.campos_sel.setSizePolicy(sizePolicy)
font = QtGui.QFont()
font.setBold(False)
font.setWeight(50)
font.setKerning(True)
font.setStyleStrategy(QtGui.QFont.PreferDefault)
self.campos_sel.setFont(font)
self.campos_sel.setMouseTracking(True)
self.campos_sel.setTabletTracking(True)
self.campos_sel.setFocusPolicy(QtCore.Qt.NoFocus)
self.campos_sel.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)
self.campos_sel.setAcceptDrops(True)
self.campos_sel.setToolTip("")
self.campos_sel.setLayoutDirection(QtCore.Qt.LeftToRight)
self.campos_sel.setAutoFillBackground(True)
self.campos_sel.setFrameShape(QtWidgets.QFrame.Box)
self.campos_sel.setLineWidth(1)
self.campos_sel.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents)
self.campos_sel.setAutoScroll(True)
self.campos_sel.setAutoScrollMargin(16)
self.campos_sel.setTabKeyNavigation(True)
self.campos_sel.setProperty("showDropIndicator", True)
self.campos_sel.setDragEnabled(True)
self.campos_sel.setAlternatingRowColors(False)
self.campos_sel.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
self.campos_sel.setTextElideMode(QtCore.Qt.ElideMiddle)
self.campos_sel.setShowGrid(False)
self.campos_sel.setGridStyle(QtCore.Qt.NoPen)
self.campos_sel.setWordWrap(True)
self.campos_sel.setCornerButtonEnabled(False)
self.campos_sel.setColumnCount(4)
self.campos_sel.setProperty("centrarTexto", True)
self.campos_sel.setObjectName("campos_sel")
self.campos_sel.setRowCount(0)
item = QtWidgets.QTableWidgetItem()
self.campos_sel.setHorizontalHeaderItem(0, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel.setHorizontalHeaderItem(1, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel.setHorizontalHeaderItem(2, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel.setHorizontalHeaderItem(3, item)
self.campos_sel.horizontalHeader().setVisible(True)
self.campos_sel.horizontalHeader().setCascadingSectionResizes(False)
self.campos_sel.horizontalHeader().setDefaultSectionSize(69)
self.campos_sel.horizontalHeader().setMinimumSectionSize(20)
self.campos_sel.horizontalHeader().setSortIndicatorShown(True)
self.campos_sel.horizontalHeader().setStretchLastSection(True)
self.campos_sel.verticalHeader().setVisible(False)
self.campos_sel.verticalHeader().setCascadingSectionResizes(False)
self.campos_sel.verticalHeader().setDefaultSectionSize(15)
self.campos_sel.verticalHeader().setHighlightSections(True)
self.campos_sel.verticalHeader().setMinimumSectionSize(10)
self.campos_sel.verticalHeader().setSortIndicatorShown(True)
```

```
self.campos_sel_2 = QtWidgets.QTableWidget(self.groupBox)
self.campos_sel_2.setEnabled(True)
self.campos_sel_2.setGeometry(QtCore.QRect(210, 150, 291, 201))
self.campos_sel_2.setMouseTracking(True)
self.campos_sel_2.setTabletTracking(True)
self.campos_sel_2.setFocusPolicy(QtCore.Qt.NoFocus)
self.campos_sel_2.setAcceptDrops(True)
self.campos_sel_2.setLayoutDirection(QtCore.Qt.LeftToRight)
self.campos_sel_2.setAutoFillBackground(True)
self.campos_sel_2.setFrameShape(QtWidgets.QFrame.Box)
self.campos_sel_2.setLineWidth(1)
self.campos_sel_2.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents)
self.campos_sel_2.setAutoScroll(True)
self.campos_sel_2.setDragEnabled(True)
self.campos_sel_2.setAlternatingRowColors(False)
self.campos_sel_2.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
self.campos_sel_2.setShowGrid(False)
self.campos_sel_2.setGridStyle(QtCore.Qt.SolidLine)
self.campos_sel_2.setWordWrap(True)
self.campos_sel_2.setCornerButtonEnabled(False)
self.campos_sel_2.setColumnCount(4)
self.campos_sel_2.setProperty("centrarTexto", True)
self.campos_sel_2.setObjectName("campos_sel_2")
self.campos_sel_2.setRowCount(0)
item = QtWidgets.QTableWidgetItem()
self.campos_sel_2.setHorizontalHeaderItem(0, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel_2.setHorizontalHeaderItem(1, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel_2.setHorizontalHeaderItem(2, item)
item = QtWidgets.QTableWidgetItem()
self.campos_sel_2.setHorizontalHeaderItem(3, item)
self.campos_sel_2.horizontalHeader().setVisible(True)
self.campos_sel_2.horizontalHeader().setCascadingSectionResizes(False)
self.campos_sel_2.horizontalHeader().setDefaultSectionSize(69)
self.campos_sel_2.horizontalHeader().setMinimumSectionSize(20)
self.campos_sel_2.horizontalHeader().setSortIndicatorShown(True)
self.campos_sel_2.horizontalHeader().setStretchLastSection(True)
self.campos_sel_2.verticalHeader().setVisible(False)
self.campos_sel_2.verticalHeader().setCascadingSectionResizes(False)
self.campos_sel_2.verticalHeader().setDefaultSectionSize(15)
self.campos_sel_2.verticalHeader().setHighlightSections(True)
self.campos_sel_2.verticalHeader().setSortIndicatorShown(True)
self.button_Add_2 = QtWidgets.QPushButton(self.groupBox)
self.button_Add_2.setGeometry(QtCore.QRect(170, 150, 31, 31))
self.button_Add_2.setObjectName("button_Add_2")
self.buttonDel_2 = QtWidgets.QPushButton(self.groupBox)
self.buttonDel_2.setGeometry(QtCore.QRect(170, 190, 31, 31))
self.buttonDel_2.setObjectName("buttonDel_2")
self.label = QtWidgets.QLabel(self.groupBox)
self.label.setGeometry(QtCore.QRect(11, 30, 121, 16))
font = QtGui.QFont()
font.setPointSize(9)
self.label.setFont(font)
self.label.setObjectName("label")
self.label_2 = QtWidgets.QLabel(self.groupBox)
self.label_2.setGeometry(QtCore.QRect(212, 30, 161, 16))
font = QtGui.QFont()
font.setPointSize(9)
self.label_2.setFont(font)
self.label_2.setObjectName("label_2")
self.label_3 = QtWidgets.QLabel(self.groupBox)
self.label_3.setGeometry(QtCore.QRect(210, 130, 161, 16))
font = QtGui.QFont()
font.setPointSize(9)
self.label_3.setFont(font)
self.label_3.setObjectName("label_3")
self.tabWidget.addTab(self.atributos, "")
self.Clases = QtWidgets.QWidget()
self.Clases.setObjectName("Clases")
self.groupBox_2 = QtWidgets.QGroupBox(self.Clases)
self.groupBox_2.setGeometry(QtCore.QRect(10, 40, 141, 271))
self.groupBox_2.setObjectName("groupBox_2")
self.tableClasesCC = QtWidgets.QTableWidget(self.groupBox_2)
self.tableClasesCC.setGeometry(QtCore.QRect(10, 20, 121, 211))
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.tableClasesCC.sizePolicy().hasHeightForWidth())
self.tableClasesCC.setSizePolicy(sizePolicy)
self.tableClasesCC.setFrameShape(QtWidgets.QFrame.Box)
self.tableClasesCC.setSelectionMode(QtWidgets.QAbstractItemView.ExtendedSelection)
self.tableClasesCC.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
self.tableClasesCC.setObjectName("tableClasesCC")
self.tableClasesCC.setColumnCount(1)
self.tableClasesCC.setRowCount(0)
item = QtWidgets.QTableWidgetItem()
self.tableClasesCC.setHorizontalHeaderItem(0, item)
self.tableClasesCC.horizontalHeader().setVisible(True)
self.tableClasesCC.horizontalHeader().setCascadingSectionResizes(False)
self.tableClasesCC.horizontalHeader().setDefaultSectionSize(90)
```

```
self.tableClasesCC.horizontalHeader().setHighlightSections(True)
self.tableClasesCC.horizontalHeader().setMinimumSectionSize(5)
self.tableClasesCC.horizontalHeader().setStretchLastSection(True)
self.tableClasesCC.verticalHeader().setVisible(False)
self.tableClasesCC.verticalHeader().setDefaultSectionSize(15)
self.tableClasesCC.verticalHeader().setMinimumSectionSize(10)
self.btnVerClasesCC = Clase_Btn_Agregar_Clases.BtnAgregarClases("Circulo central",self.groupBox_2)
self.btnVerClasesCC.setGeometry(QtCore.QRect(10, 240, 121, 23))
self.btnVerClasesCC.setObjectName("btnVerClasesCC")
self.groupBox_3 = QtWidgets.QGroupBox(self.Clases)
self.groupBox_3.setGeometry(QtCore.QRect(164, 40, 141, 271))
self.groupBox_3.setObjectName("groupBox_3")
self.tableClasesSC = QtWidgets.QTableWidget(self.groupBox_3)
self.tableClasesSC.setGeometry(QtCore.QRect(10, 20, 121, 211))
self.tableClasesSC.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
self.tableClasesSC.setObjectName("tableClasesSC")
self.tableClasesSC.setColumnCount(1)
self.tableClasesSC.setRowCount(0)
item = QtWidgets.QTableWidgetItem()
self.tableClasesSC.setHorizontalHeaderItem(0, item)
self.tableClasesSC.horizontalHeader().setVisible(True)
self.tableClasesSC.horizontalHeader().setDefaultSectionSize(90)
self.tableClasesSC.horizontalHeader().setMinimumSectionSize(5)
self.tableClasesSC.horizontalHeader().setStretchLastSection(True)
self.tableClasesSC.verticalHeader().setVisible(False)
self.tableClasesSC.verticalHeader().setDefaultSectionSize(15)
self.tableClasesSC.verticalHeader().setMinimumSectionSize(15)
self.btnVerClasesSC = Clase_Btn_Agregar_Clases.BtnAgregarClases("Segmentos de diagrama",self.groupBox_3)
self.btnVerClasesSC.setGeometry(QtCore.QRect(10, 240, 121, 23))
self.btnVerClasesSC.setObjectName("btnVerClasesSC")
self.groupBox_5 = QtWidgets.QGroupBox(self.Clases)
self.groupBox_5.setGeometry(QtCore.QRect(320, 40, 221, 271))
self.groupBox_5.setObjectName("groupBox_5")
self.tableClasesED = QtWidgets.QTableWidget(self.groupBox_5)
self.tableClasesED.setGeometry(QtCore.QRect(10, 20, 201, 211))
self.tableClasesED.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)
self.tableClasesED.setObjectName("tableClasesED")
self.tableClasesED.setColumnCount(2)
self.tableClasesED.setRowCount(0)
item = QtWidgets.QTableWidgetItem()
self.tableClasesED.setHorizontalHeaderItem(0, item)
item = QtWidgets.QTableWidgetItem()
self.tableClasesED.setHorizontalHeaderItem(1, item)
self.tableClasesED.horizontalHeader().setVisible(True)
self.tableClasesED.horizontalHeader().setDefaultSectionSize(90)
self.tableClasesED.horizontalHeader().setMinimumSectionSize(5)
self.tableClasesED.horizontalHeader().setStretchLastSection(True)
self.tableClasesED.verticalHeader().setVisible(False)
self.tableClasesED.verticalHeader().setDefaultSectionSize(15)
self.tableClasesED.verticalHeader().setMinimumSectionSize(15)
self.tabWidget.addTab(self.Clases, "")
self.renderizado = QtWidgets.QWidget()
self.renderizado.setObjectName("renderizado")
self.groupBox_4 = QtWidgets.QGroupBox(self.renderizado)
self.groupBox_4.setGeometry(QtCore.QRect(20, 20, 511, 361))
self.groupBox_4.setObjectName("groupBox_4")
self.label_7 = QtWidgets.QLabel(self.groupBox_4)
self.label_7.setGeometry(QtCore.QRect(10, 40, 91, 16))
self.label_7.setObjectName("label_7")
self.label_8 = QtWidgets.QLabel(self.groupBox_4)
self.label_8.setGeometry(QtCore.QRect(10, 100, 91, 16))
self.label_8.setObjectName("label_8")
self.colorLinea = QgsColorButton(self.groupBox_4)
self.colorLinea.setGeometry(QtCore.QRect(140, 100, 101, 21))
self.colorLinea.setAllowOpacity(False)
self.colorLinea.setBehavior(QgsColorButton.ShowDialog)
self.colorLinea.setDefaultColor(QtGui.QColor(0, 0, 0))
self.colorLinea.setObjectName("colorLinea")
self.label_9 = QtWidgets.QLabel(self.groupBox_4)
self.label_9.setGeometry(QtCore.QRect(290, 100, 91, 16))
self.label_9.setObjectName("label_9")
self.anchoLinea = QtWidgets.QDoubleSpinBox(self.groupBox_4)
self.anchoLinea.setGeometry(QtCore.QRect(390, 100, 101, 21))
self.anchoLinea.setObjectName("anchoLinea")
self.label_10 = QtWidgets.QLabel(self.groupBox_4)
self.label_10.setGeometry(QtCore.QRect(290, 160, 91, 16))
self.label_10.setObjectName("label_10")
self.anguloInicio = QtWidgets.QComboBox(self.groupBox_4)
self.anguloInicio.setGeometry(QtCore.QRect(390, 160, 101, 21))
self.anguloInicio.setObjectName("anguloInicio")
self.anguloInicio.addItem("")
self.anguloInicio.addItem("")
self.anguloInicio.addItem("")
self.anguloInicio.addItem("")
self.transparencia = QgsOpacityWidget(self.groupBox_4)
self.transparencia.setGeometry(QtCore.QRect(140, 40, 351, 27))
self.transparencia.setObjectName("transparencia")
self.distanciaOb = QtWidgets.QDoubleSpinBox(self.groupBox_4)
self.distanciaOb.setGeometry(QtCore.QRect(140, 160, 101, 21))
self.distanciaOb.setObjectName("distanciaOb")
self.label_12 = QtWidgets.QLabel(self.groupBox_4)
```

```
        self.label_12.setGeometry(QtCore.QRect(10, 160, 111, 16))
        self.label_12.setObjectName("label_12")
        self.groupBox_6 = QtWidgets.QGroupBox(self.groupBox_4)
        self.groupBox_6.setGeometry(QtCore.QRect(10, 220, 251, 121))
        self.groupBox_6.setObjectName("groupBox_6")
        self.ajusteTamT2 = QtWidgets.QDoubleSpinBox(self.groupBox_6)
        self.ajusteTamT2.setGeometry(QtCore.QRect(130, 80, 101, 21))
        self.ajusteTamT2.setObjectName("ajusteTamT2")
        self.ajusteTamT1 = QtWidgets.QDoubleSpinBox(self.groupBox_6)
        self.ajusteTamT1.setGeometry(QtCore.QRect(130, 40, 101, 21))
        self.ajusteTamT1.setObjectName("ajusteTamT1")
        self.label_13 = QtWidgets.QLabel(self.groupBox_6)
        self.label_13.setGeometry(QtCore.QRect(10, 80, 101, 21))
        self.label_13.setObjectName("label_13")
        self.label_11 = QtWidgets.QLabel(self.groupBox_6)
        self.label_11.setGeometry(QtCore.QRect(10, 40, 121, 16))
        self.label_11.setObjectName("label_11")
        self.groupBox_7 = QtWidgets.QGroupBox(self.groupBox_4)
        self.groupBox_7.setGeometry(QtCore.QRect(280, 220, 221, 121))
        self.groupBox_7.setObjectName("groupBox_7")
        self.maximTamT2 = QtWidgets.QDoubleSpinBox(self.groupBox_7)
        self.maximTamT2.setGeometry(QtCore.QRect(110, 80, 101, 21))
        self.maximTamT2.setObjectName("maximTamT2")
        self.label_15 = QtWidgets.QLabel(self.groupBox_7)
        self.label_15.setGeometry(QtCore.QRect(10, 80, 101, 21))
        self.label_15.setObjectName("label_15")
        self.maximTamT1 = QtWidgets.QDoubleSpinBox(self.groupBox_7)
        self.maximTamT1.setGeometry(QtCore.QRect(110, 40, 101, 21))
        self.maximTamT1.setObjectName("maximTamT1")
        self.label_14 = QtWidgets.QLabel(self.groupBox_7)
        self.label_14.setGeometry(QtCore.QRect(10, 40, 101, 16))
        self.label_14.setObjectName("label_14")
        self.tabWidget.addTab(self.renderizado, "")
        self.opciones = QtWidgets.QWidget()
        self.opciones.setObjectName("opciones")
        self.tabWidget.addTab(self.opciones, "")
        self.etiquetas = QtWidgets.QWidget()
        self.etiquetas.setObjectName("etiquetas")
        self.tabWidget.addTab(self.etiquetas, "")
        self.buttonCancelar = QtWidgets.QPushButton(self.centralwidget)
        self.buttonCancelar.setGeometry(QtCore.QRect(435, 470, 75, 23))
        self.buttonCancelar.setObjectName("buttonCancelar")
        self.buttonAceptar_3 = QtWidgets.QPushButton(self.centralwidget)
        self.buttonAceptar_3.setGeometry(QtCore.QRect(515, 470, 75, 23))
        self.buttonAceptar_3.setObjectName("buttonAceptar_3")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 619, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)

        self.retranslateUi(MainWindow)
        self.tabWidget.setCurrentIndex(0)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.diagramSel.setItemText(0, _translate("MainWindow", "Cartodiagrama estructurado"))
        self.diagramSel.setItemText(1, _translate("MainWindow", "Tipograma lineal dinamico"))
        self.diagramSel.setItemText(2, _translate("MainWindow", "Cartodiagrama de celdas"))
        self.buttonAceptar.setText(_translate("MainWindow", "Aceptar"))
        self.tabWidget.setToolTip(_translate("MainWindow", "<html><head/><body><p>jsjdjdsd</p></body></html>"))
        self.groupBox.setTitle(_translate("MainWindow", "Attributos"))
        self.campos_capa.setSortingEnabled(True)
        item = self.campos_capa.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Atributos"))
        self.button_Add.setText(_translate("MainWindow", "Add"))
        self.buttonDel.setText(_translate("MainWindow", "Del"))
        self.campos_sel.setSortingEnabled(True)
        item = self.campos_sel.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Atributos"))
        item = self.campos_sel.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Color"))
        item = self.campos_sel.horizontalHeaderItem(2)
        item.setText(_translate("MainWindow", "Etiqueta"))
        item = self.campos_sel.horizontalHeaderItem(3)
        item.setText(_translate("MainWindow", "Sectores"))
        self.campos_sel_2.setSortingEnabled(True)
        item = self.campos_sel_2.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Atributos"))
        item = self.campos_sel_2.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Color"))
        item = self.campos_sel_2.horizontalHeaderItem(2)
        item.setText(_translate("MainWindow", "Etiqueta"))
        item = self.campos_sel_2.horizontalHeaderItem(3)
        item.setText(_translate("MainWindow", "Sectores"))
        self.button_Add_2.setText(_translate("MainWindow", "Add"))
        self.buttonDel_2.setText(_translate("MainWindow", "Del"))
        self.label.setText(_translate("MainWindow", "Atributos Disponibles"))
        self.label_2.setText(_translate("MainWindow", "Atributo(s) (circulo central)"))
```

```
        self.label_3.setText(_translate("MainWindow", "Atributos (Segmentos y Ejes)"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.atributos), _translate("MainWindow", "Atributos"))
        self.groupBox_2.setTitle(_translate("MainWindow", "Circulo central"))
        item = self.tableClasesCC.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Atributos"))
        self.btnVerClasesCC.setText(_translate("MainWindow", "Ver clases"))
        self.groupBox_3.setTitle(_translate("MainWindow", "Segmentos de diagrama"))
        item = self.tableClasesSC.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Atributos"))
        self.btnVerClasesSC.setText(_translate("MainWindow", "Ver clases"))
        self.groupBox_5.setTitle(_translate("MainWindow", "Ejes de diagrama"))
        item = self.tableClasesED.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Atributos"))
        item = self.tableClasesED.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Clases"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.Clases), _translate("MainWindow", "Clases"))
        self.groupBox_4.setTitle(_translate("MainWindow", "Renderizado y tamaño"))
        self.label_7.setText(_translate("MainWindow", "Transparencia"))
        self.label_8.setText(_translate("MainWindow", "Color de linea"))
        self.label_9.setText(_translate("MainWindow", "Ancho de linea"))
        self.label_10.setText(_translate("MainWindow", "Angulo de inicio"))
        self.anguloInicio.setItemText(0, _translate("MainWindow", "Arriba"))
        self.anguloInicio.setItemText(1, _translate("MainWindow", "Derecha"))
        self.anguloInicio.setItemText(2, _translate("MainWindow", "Abajo"))
        self.anguloInicio.setItemText(3, _translate("MainWindow", "Izquierda"))
        self.label_12.setText(_translate("MainWindow", "Tamaño de separacion"))
        self.groupBox_6.setTitle(_translate("MainWindow", "Ajuste de tamaño para clases"))
        self.label_13.setText(_translate("MainWindow", "Segmentos y ejes"))
        self.label_11.setText(_translate("MainWindow", "Circulo central"))
        self.groupBox_7.setTitle(_translate("MainWindow", "Tamaño maximo para escala continua"))
        self.label_15.setText(_translate("MainWindow", "Segmentos y ejes"))
        self.label_14.setText(_translate("MainWindow", "Circulo central"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.renderizado), _translate("MainWindow", "Renderizado"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.opciones), _translate("MainWindow", "Opciones"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.etiquetas), _translate("MainWindow", "Etiquetas"))
        self.buttonCancelar.setText(_translate("MainWindow", "Cancelar"))
        self.buttonAceptar_3.setText(_translate("MainWindow", "Aplicar"))

from qgscolorbutton import QgsColorButton
from qgsopacitywidget import QgsOpacityWidget

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

12.5 VentanaIntervaloClases

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.setFixedSize(265, 169)
        self.label = QtWidgets.QLabel(Dialog)
        self.label.setGeometry(QtCore.QRect(20, 30, 81, 16))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(Dialog)
        self.label_2.setGeometry(QtCore.QRect(20, 80, 71, 16))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.valorInicial = QtWidgets.QLineEdit(Dialog)
        self.valorInicial.setGeometry(QtCore.QRect(110, 30, 131, 20))
        self.valorInicial.setObjectName("valorInicial")
        self.ValorFinal = QtWidgets.QLineEdit(Dialog)
        self.ValorFinal.setGeometry(QtCore.QRect(110, 80, 131, 20))
        self.ValorFinal.setObjectName("ValorFinal")
        self.buttonBox = QtWidgets.QDialogButtonBox(Dialog)
        self.buttonBox.setGeometry(QtCore.QRect(110, 130, 131, 23))
        self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok)
        self.buttonBox.setObjectName("buttonBox")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.label.setText(_translate("Dialog", "Valor inicial"))
```

```
        self.label_2.setText(_translate("Dialog", "Valor final"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```