Reporte de Laboratorio X
Implementación de una aplicación móvil para detección de distraídos

Coyoy López Mario*
Ledezma Donjuan Daniel Armando *
Olivares Rodríguez Brayan *
Torres Colorado Juan Daniel*
*Ingeniería en Tecnologías de la Información
Universidad Politécnica de Victoria

***Resumen*— En este proyecto se desarrolló un programa en C++ utilizando la biblioteca Qt para resolver el problema del cierre convexo (convex hull) utilizando varios algoritmos: Andrew's Monotone Chain, Graham Scan y Shamos. El programa permite al usuario crear un conjunto de puntos, editar estos puntos y luego aplicar los algoritmos mencionados para encontrar el cierre convexo del conjunto de puntos dado. La interfaz proporciona opciones intuitivas para ingresar y manipular los puntos, así como para seleccionar el algoritmo deseado. Se utilizan estructuras de datos como vectores y contenedores para almacenar y manipular los puntos, garantizando una ejecución eficiente de los algoritmos. El programa ofrece una experiencia interactiva y amigable para el usuario, facilitando la comprensión y la interacción con el proceso de cálculo del cierre convexo.**

I. Introducción

En el presente documento del proyecto de nombre "Algoritmo Convex Hull" se detalla el trabajo realizado y asignado como proyecto en equipo de la asignatura de Lenguajes y Autómatas en el lenguaje de programación C++ [1]. El propósito de este proyecto tiene como motivo principal comprender los algoritmos de una envolvente convexa de manera visual, la cuál, cuente con una interfaz interactiva[2].

La *Convex Hull* (Envolvente Convexa) [3] de un conjunto de puntos en un espacio euclidiano [4] es el polígono convexo [5] más pequeño que encierra todos los puntos. Este tipo de herramienta resulta fundamental tiene diversas aplicaciones (entre otros más) en campos como la visión por computadora, geometría computacional, el procesamiento de imágenes o la planificación de rutas. Como su nombre dicta, los algoritmos de la envolvente convexa buscan encontrar la envolvente convexa de un conjunto de puntos para finalmente encerrarlas en un polígono convexo, algunos de estos algoritmos que se harán de uso son:

- **Andrew's Monotone Chain** [6]: El algoritmo de Andrew se enfoca primeramente en ordenar los puntos y luego calcular las envolventes superior e inferior. Después, los puntos se ordenarán con respecto a las coordenadas x (respecto a y en caso de empatar con x), luego encontraremos al punto más a la izquierda, rotamos en sentido horario y encontramos el siguiente punto. Por último, repetiremos el paso hasta llegar al punto más a la derecha, giramos en sentido horario y encontramos la envolvente inferior.
- **Graham Scan** [7]: Este algoritmo comienza encontrando el punto con la coordenada más pequeña, dicho punto siempre está en la envolvente convexa. Después, ordenamos los puntos restantes por su ángulo polar con respecto al punto inicial. Luego, agregaremos puntos iterativamente a la envolvente convexa. En cada paso, se comprueba si los dos últimos puntos forman un giro a la derecha y de ser así, se elimina el último punto, caso contrario, se añadirá. Este algoritmo terminará cuando se haya agregado todos los puntos a la envolvente convexa.
- **Shamos Hoey** [8]: Su propósito difiere a las anteriores mencionadas, ya que esta encuentra los pares de puntos más cercanos en un conjunto de puntos. El algoritmo comienza dividiendo el conjunto de puntos en dos subconjuntos más pequeños, divididos por una línea vertical en el plano. Esta línea se elige de modo que divida el conjunto de puntos en dos subconjuntos casi iguales. Luego, el algoritmo se aplica a cada uno de los subconjuntos más pequeños para encontrar los pares de puntos más cercanos en cada subconjunto. Después se fusiona las soluciones para encontrar los pares de puntos más cercanos en todo el conjunto.

II. Desarrollo Experimental

Para comenzar con el desarrollo del proyecto, hay que definir la importancia del algoritmo [9] Convex Hull. De acuerdo a la definición del algoritmo Convex Hull, se trata de una estructura de gran importancia en la geometría computacional, dado que se trata de una gran herramienta por sí sola, y además es útil para construir otras estructuras como diagramas de Voronoi e incluso aplicación de análisis de imágenes.

La visualización del algoritmo se puede determinar con la siguiente analogía de clavos: Si tuviéramos una superficie, como una tabla, en la que se encuentran diversos clavos (puntos) sin un orden en específico, tomamos una liga o banda elástica y la estiramos hasta que todos los clavos alrededor de

esta y lo soltamos, la liga tomará la forma de los clavos que minimice su longitud. El área dentro de la liga es el Convex Hull.

Después de entender la importancia de este algoritmo, en este trabajo se llevó a cabo la implementación de dicho algoritmo de manera gráfica en una aplicación en C++ usando QT5 [10]. El primer paso para la realización de este proyecto, como se aprecia en los párrafos anteriores, fue abordar el tema mediante una profunda investigación, de la cual se determinó la forma de trabajar para dividir tareas entre el equipo. Dado que el algoritmo anterior mencionado es complejo, se requieren implementar diversas variantes para que el proyecto estuviese completo.

El siguiente algoritmo es el denominado Graham Scan que se expone en el siguiente pseudocódigo, en el cual se muestran los pasos a seguir para obtener el resultado deseado.

Algorithm 1 Convex Hull - Graham Scan

Select the point with minimum y
Sort all points in CCW order $p_0, p_1, \ldots, p_n$
$S = p_0, p_1$
for $i = 2$ to n **do**
  while $|S| > 2$ **and** p_i is to the right of $S_{|S|-2}, S_{|S|-1}$ **do**
    S.pop()
  end while
  S.push(p_i)
end for

Continuando con el desarrollo del proyecto, se realizó el desarrollo e implementación del algoritmo Andrew's Chain, el cual es otro algoritmo bastante usado para encontrar el Convex Hull de un conjunto de puntos en el plano. El algoritmo se muestra a continuación.

Algorithm 2 Convex Hull - Andrew's Chain

Sort all points by their x-coordinate
Initialize empty lists $lower\_hull$ and $upper\_hull$
for each point p in sorted points **do**
  while $|lower\_hull| \geq 2$ and orientation($lower\_hull[|lower\_hull| - 2]$, $lower\_hull[|lower\_hull| - 1]$, p) ≤ 0 **do**
    $lower\_hull$.pop()
  end while
  $lower\_hull$.append(p)
end for
for each point p in sorted points in reverse order **do**
  while $|upper\_hull| \geq 2$ and orientation($upper\_hull[|upper\_hull| - 2]$, $upper\_hull[|upper\_hull| - 1]$, p) ≤ 0 **do**
    $upper\_hull$.pop()
  end while
  $upper\_hull$.append(p)
end for
$convex\_hull = lower\_hull[:-1] + upper\_hull[:-1]$

El siguiente algoritmo es el de Shamos, el cual también es conocido como el algoritmo del escaneo de líneas, es un enfoque basado en el escaneo de una línea vertical a través del conjunto de puntos ordenados por coordenada x. A medida que esta línea vertical se desplaza de izquierda a derecha, se mantiene un conjunto de puntos activos que están .[en]cima"de la línea vertical. Estos puntos activos son aquellos que pueden contribuir a la envolvente convexa en el momento dado. El algoritmo de Shamos es el siguiente.

Algorithm 3 Shamos - GetAllAntiPodalPairs($p[1..n]$)

$i = 1$
$j = 2$
while angle$(i, j) < \pi$ **do**
  $j++$
end while
yield i, j
$current = i$
while $j \neq n$ **do**
  if angle$(current, i+1) \leq$ angle$(current, j+1)$ **then**
    $j++$
    $current = j$
  else
    $i++$
    $current = i$
  end if
  yield i, j
  if angle$(current, i+1) =$ angle$(current, j+1)$ **then**
    yield $i+1, j$
    yield $i, j+1$
    yield $i+1, j+1$
    if $current = i$ **then**
      $j++$
    else
      $i++$
    end if
  end if
end while

Después de entender el funcionamiento de los algoritmos anteriores, el siguiente paso realizado fue su implementación. La cual se puede observar en la figura 8, donde se muestra un diagrama de clases del proyecto y la relación entre las clases del progreso final del proyecto. El programa hace uso de cuatro clases. La más importante es la clase MainWindow, dado que en esta es donde ocurre la ejecución de la aplicación como tal, además de realizar todos los algoritmos y su ejecución en tiempo real de acuerdo a su procedimientos. Algunas funciones importantes son AndrewChain(), drawAndrew() y andrewsChain() que se encarga del algoritmo Andrew's Chain, el cual como se menció anteriormente es una variación del algoritmo Convex Hull. Además, se implementaron los otros dos algoritmos mencionados anteriormente: Graham Scan y el algoritmo de Shamos. El algoritmo de Graham Scan, implementado en la función grahamScan(), utiliza una pila para construir la envolvente convexa a partir de un conjunto

de puntos. Comienza encontrando el punto más bajo y más a la izquierda (el punto de partida) y luego ordena los demás puntos según el ángulo polar que forman con este punto. Luego, procesa estos puntos de manera secuencial para formar el Convex Hull con la ayuda de la función drawCirclesGH(). Por otro lado, el algoritmo de Shamos, implementado en la función shamosAlgorithm() fue creado con la técnica de ”divide y conquistarás”. Este algoritmo divide recursivamente el conjunto de puntos en dos partes, encuentra las envolventes convexas de cada parte y luego las combina para obtener la envolvente convexa global con la función drawCirclesShamos().

III. Resultados

1. **Interfaz de Inicio del Programa:** La interfaz de inicio del programa se compone de una ventana principal que presenta seis botones claramente etiquetados para facilitar la navegación y el acceso a las distintas funciones del programa. Estos botones son:
   - Create: Este botón permite al usuario generar círculos de manera aleatoria para utilizar en las tres funciones principales del programa: Andrew Chain, Ch Graham y Shamos.
   - Edit Points: Proporciona al usuario la capacidad de editar los puntos generados previamente, permitiendo ajustes personalizados según las necesidades del proyecto.
   - Example Point: Ofrece la posibilidad de generar puntos de ejemplo que sirven para ilustrar y entender mejor el funcionamiento de las funciones del programa.
   - Andrew Chain: Al hacer clic en este botón, se activa la función Andrew Chain del programa, que realiza una serie de operaciones específicas sobre los puntos generados, produciendo un resultado visual determinado.
   - Ch Graham: Al seleccionar este botón, se accede a la función Ch Graham del programa, la cual ejecuta un algoritmo específico sobre los puntos dados para realizar una tarea particular, proporcionando un resultado visual correspondiente.
   - Shamos: Al hacer clic aquí, se activa la función Shamos del programa, que realiza una serie de operaciones algorítmicas sobre los puntos generados, ofreciendo un resultado visual único y relevante.

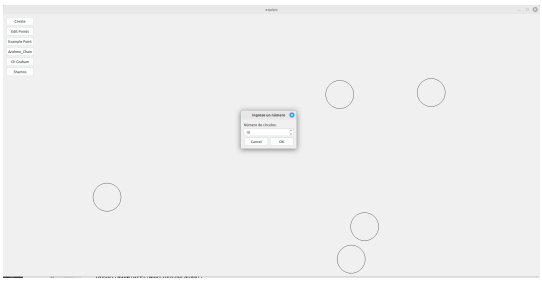

Figura 1: Interfaz de Edición de Puntos

2. **Interfaz de Inicio del Programa:** La Figura 2 muestra la interfaz de edición de puntos del programa. Al presionar el botón .Edit Points”, se abre una ventana emergente que permite al usuario realizar modificaciones específicas en los puntos generados previamente. Esta ventana presenta las siguientes características:
   - Al hacer clic en cualquier posición dentro de la ventana emergente, se crea un nuevo círculo en esa ubicación, lo que proporciona al usuario la capacidad de añadir puntos adicionales de manera intuitiva y precisa. La ventana de edición incluye tres botones adicionales para mejorar la experiencia del usuario:
     - **Cancelar:** Este botón permite al usuario salir de la ventana emergente sin guardar los cambios realizados, brindando una opción para abortar la operación actual si es necesario.
     - **Borrar Dibujo:** Al seleccionar este botón, se borran todos los círculos dibujados en la ventana de edición, lo que ofrece una manera rápida y conveniente de eliminar los puntos existentes y comenzar de nuevo si es necesario.
     - **Done:** Al hacer clic en este botón, se confirman y guardan los cambios realizados en la ventana de edición de puntos, permitiendo al usuario finalizar el proceso de edición y regresar a la interfaz principal del programa con los ajustes actualizados.

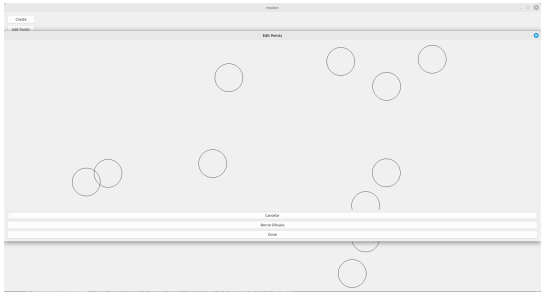

Figura 2: Interfaz de Edición de Puntos

3. **Ejemplos de Figuras Generadas:** La Figura 3 muestra cuatro ejemplos representativos de figuras geométricas generadas a partir de círculos. Estos ejemplos ilustran las siguientes figuras:
   - **Cuadrado (Square):** Se muestra un ejemplo de un cuadrado generado mediante la disposición estratégica de varios círculos, demostrando cómo la combinación adecuada de formas básicas puede crear figuras más complejas y reconocibles.
   - **Pentágono (Pentagon):** En este ejemplo, se presenta un pentágono formado por la intersección y superposición de múltiples círculos, ilustrando cómo esta técnica puede emplearse para construir polígonos regulares de varios lados.

- **Creciente (Crescent):** Se exhibe un ejemplo de una creciente o luna creciente, modelada mediante la disposición cuidadosa de círculos para simular la curvatura característica de este tipo de figura, mostrando así la versatilidad del enfoque utilizado.
- **Círculo (Circle):** Por último, se incluye un ejemplo simple de un círculo formado por la superposición de varios círculos concéntricos, destacando cómo la técnica puede utilizarse para representar formas básicas con precisión y detalle.

Estos ejemplos proporcionan una visualización clara y práctica de las capacidades del programa al generar figuras complejas a partir de la combinación y manipulación de elementos simples, demostrando así su utilidad y versatilidad en la creación de diseños geométricos variados.

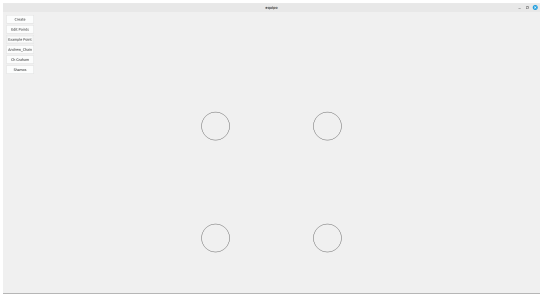

Figura 3: Ejemplo de Figura Generada (Square)

4. **Funciones Aplicadas a los Puntos:** La Figura 4 ilustra cómo las funciones Andrew Chain 7 , Ch Graham 5 o Shamos 6 del programa se aplican a los puntos presentes en la pantalla en ese momento. Estos puntos pueden ser generados de tres maneras diferentes:

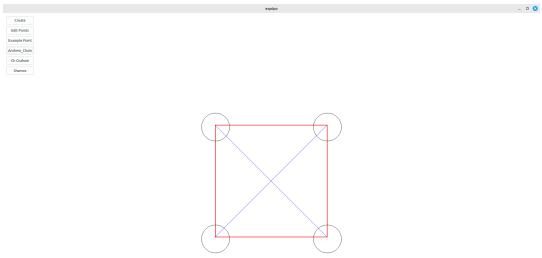

Figura 4: Funciones Aplicadas a los Puntos

- **Ejemplos Predefinidos:** Los puntos pueden ser ejemplos predefinidos proporcionados por el programa, que representan diversas figuras geométricas como cuadrados, pentágonos, crecientes, entre otros.
- **Generados Aleatoriamente:** También es posible que los puntos sean generados aleatoriamente por el programa, lo que permite una variedad de formas y distribuciones para ser procesadas por las funciones seleccionadas.
- **Modificados por el Usuario:** Los puntos pueden ser creados o modificados por el usuario mediante la función de edición de puntos, lo que brinda flexibilidad para personalizar y ajustar las figuras según las necesidades específicas.

Al aplicar una de las funciones mencionadas a los puntos presentes en la pantalla, se ejecuta un algoritmo específico diseñado para realizar una tarea particular, como el cálculo de la envolvente convexa (Andrew Chain), la búsqueda del punto más bajo y la ordenación angular (Ch Graham), o la búsqueda de pares de puntos más cercanos (Shamos). El resultado de estas operaciones se visualiza en la pantalla, proporcionando al usuario una representación clara y comprensible de los resultados de la función aplicada a los puntos seleccionados. Esta capacidad del programa permite explorar y analizar de manera interactiva diversas estructuras geométricas y relaciones entre puntos, facilitando la comprensión y el estudio de conceptos clave en geometría computacional.

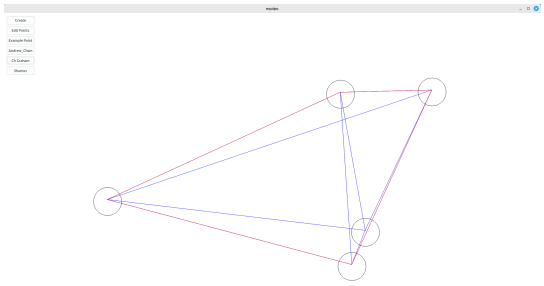

Figura 5: Funcion de Ch Graham

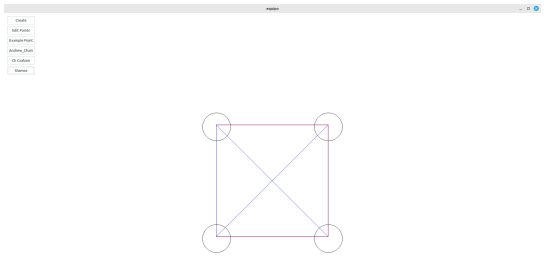

Figura 6: Funcion de Shamos

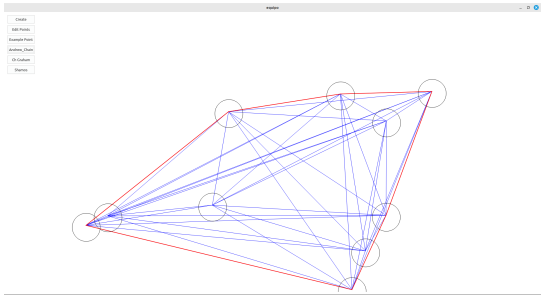

Figura 7: Funcion de Andrew Chain

IV. Conclusión

En conclusión, este proyecto ha logrado con éxito los objetivos establecidos al desarrollar una aplicación en C++ utilizando la biblioteca Qt para resolver el problema de la envolvente convexa. A través de la investigación exhaustiva, el diseño cuidadoso y la implementación efectiva de los algoritmos de Andrew's Monotone Chain, Graham Scan y Shamos, se ha creado una herramienta interactiva y amigable para el usuario.

La aplicación resulta ofrecer una interfaz intuitiva que permite a los usuarios generar conjuntos de puntos, editarlos según sea necesario y aplicar dichos algoritmos para encontrar la envolvente convexa. Los algoritmos implementados funcionan de manera eficiente y producen resultados precisos incluso para conjuntos de puntos grandes.

Por última instancia, este proyecto contribuyó significativamente tanto al crecimiento personal como en equipo, ya que a través del trabajo en equipo se llevó la paciencia, la tranquilidad, la responsabilidad, el esfuerzo y la productividad. También aportando conocimiento teórico y práctico, logrando tener un impacto significativo en diversas áreas, como la geometría computacional, la visión por computadora y el procesamiento de imágenes. Este proyecto proporciona una base sólida para abordar desafíos más avanzados y complejos en futuros trabajos y colaboraciones.

Referencias

[1] Bjarne Stroustrup. *The C++ programming language*. 2000.

[2] TechTarget Fred Churchville. *What is user interface (UI)?* https : / / www . techtarget . com / searchapparchitecture / definition / user - interface - UI. Consultado el 27-03-2024.

[3] Ask Neve Gamby & Jyrki Katajainen. "Convex-Hull Algorithms: Implementation, Testing, and Experimentation". En: *MDPI Journal* 11 (2018), págs. 2-19.

[4] Academia Lab. *Espacio euclidiano*. https://academia-lab.com/enciclopedia/espacio-euclidiano/. Consultado el 27-03-2024. 2024.

[5] Israel Aguirre. *Clasificación de polígonos: Cóncavo y convexo*. https://www.geogebra.org/m/G8dWmRJh. Consultado el 27-03-2024.

[6] Joubert & H. Leather & M. Parsons. *Parallel Computing: On the Road to Exascale*. 2016.

[7] Steven S Skiena. *The Algorithm Design Manual*. 2009.

[8] Michael Ian Shamost & Dan Hoey. *CLOSEST-POINT PROBLEMS*. 1975.

[9] GCFGlobal. *Computer Science - Algorithms*. https://edu.gcfglobal.org/en/computer-science/algorithms/1/. Consultado el 27-03-2024.

[10] Packt. *What is Qt?* https://subscription.packtpub.com/book/programming/9781788397827/1/ch01lvl1sec02/what-is-qt. Consultado el 27-03-2024.

V. ANEXOS

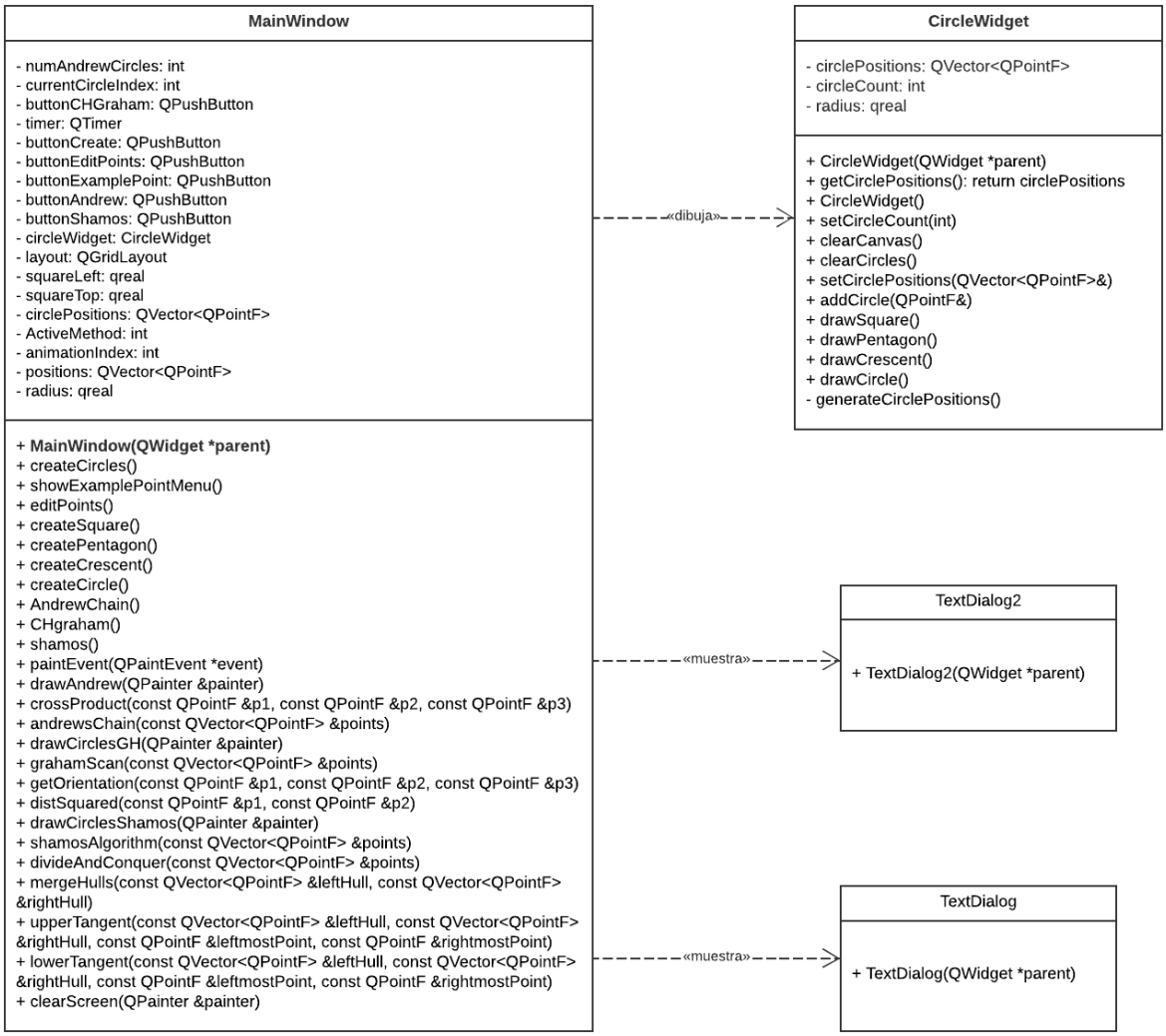

Figura 8: Diagrama de clases.