Capítulo 4

En este capítulo se explica la implementación tanto de la librería como de la aplicación, es decir el cómo todos los componentes se entrelazan entre sí y los requisitos necesarios para replicarlas.

4.1Librería

La librería se encuentra disponible para su uso y requiere únicamente de agregarla a la aplicación en la que se está trabajando. El uso de la librería requiere necesariamente la creación de un objeto de tipo ubicación, opcionalmente se puede crear uno de tipo controlador y agregar a él las distintas ubicaciones de cada jugador, existe para ello un método especial.

El controlador se encarga de cualquier ubicación, ya sea de un obstáculo, jugador o meta, soportando múltiples instancias de cualquiera de ellos. En caso que alguno de ellos cambie de ubicación el controlador detecta esto y automáticamente busca colisiones, encuentra cercanía con obstáculos o llegada de un jugador a cualquier meta. Cada uno de estos elementos, jugadores, obstáculos y metas tienen un método específico para agregarse al controlador de manera que no haya confusión en el tipo de entidad que se agrega. En una partida para múltiples jugadores a cada jugador le corresponde un objeto de tipo ubicación, de igual forma a cada obstáculo y meta.

En caso que lo que se desee es una partida para un jugador sin obstáculos ni objetivos a alcanzar únicamente es necesario asignarle un objeto de tipo ubicación y trabajar sobre él; es decir en este caso el controlador no es necesario.

Con el objetivo de conocer los valores exactos de latitud y longitud necesarios para avanzar hacia una dirección con una orientación dada para poder avanzar hacia una nueva posición es necesario recurrir a las coordenadas polares. Estas se grafican en un punto en el

sistema polar utilizando dos valores, uno de ellos es la longitud de la línea y el otro es el ángulo dado en radianes.

En WITS el ángulo está dado en grados y consiste del valor de la orientación. Por otra parte para WITS cada vez que se avance se considera el avance de una unidad, esta es π/2 multiplicado por una escala definida como 0.000080, la cual da buen ritmo a la aplicación, esto quiere decir que da al usuario la sensación que está recorriendo las calles de manera rápida. Este valor no cambia a menos que el usuario pase por una zona de tráfico.

Parte de la teoría de coordenadas polares establece que para obtener el valor de “x” y “y” se necesita la longitud de la línea y la orientación obteniendo el coseno del ángulo para “x”, la cual se convierte en latitud, siendo lo mismo para “y” donde se debe obtener el seno para obtener la longitud.

Las coordenadas obtenidas se miden en un sistema de medición de ángulos estándar, sin embargo Street View utiliza su propio sistema de medición, siendo el Norte los cero grados y aumentando en sentido de las manecillas del reloj por lo que es necesario transformar los valores obtenidos a este sistema de medición.

Para contrarrestar el giro en sentido contrario se obtiene su complemento, restándole a trescientos sesenta el ángulo, representado como Ɵ. Para que el cero se encuentre en el Norte al complemento del ángulo se le suman noventa grados y con el objetivo de evitar que esta cantidad supere los trescientos sesenta grados se hace una operación módulo trescientos sesenta. A este resultado se le aplica la operación de seno y coseno respectivamente para obtener la latitud y longitud. De esta forma las fórmulas quedan de la siguiente forma:

Latitud=sin{[(360-Ɵ)+90]%360}

Longitud=cos{[(360-Ɵ)+90]%360}

Una vez definidas las posiciones de cada jugador es necesario mostrarlas gráficamente y

para cada una de ellas agregar un componente que permita la modificación por parte del usuario.

La librería permite la creación de un gran número de aplicaciones con fines de entretenimiento; en este trabajo sólo se muestra una de ellas, sin embargo un escenario posible es una compañía de taxis que desee el control de sus unidades, que pueden funcionar como jugadores, donde cada una de ellos debe dejar a su pasajero en una ubicación determinada que representa una meta. Entre los choferes hay comunicación por radio y de esta manera pueden avisar a los demás en qué zona de la ciudad hay tráfico, lo cual se tomaría como un obstáculo. De esta manera WITS puede controlar el movimiento y facilidad de control de rutas por parte de la compañía encargada, la cual sería modelada por un objeto de tipo controlador.

4.2 Requisitos

En la plataforma de Windows es necesario contar con un emulador de GNU, como lo es minGW además de varios complementos del mismo que se enlistan a continuación: Bison, Perl, Flex, Flex++, Gperf e iconv los cuales ayudan a mostrar el mapa a cada usuario y a que los protocolos multicontacto puedan funcionar de manera correcta.

El uso de Street View en la aplicación hace que sea necesario tener instalada la última versión de Flash para poder visualizarlo por el hecho que este servicio está basado en esta herramienta. En caso que el sistema operativo de la computadora donde se realice la aplicación sea de sesenta y cuatro bits es necesario indicar al ambiente de desarrollo el lugar exacto donde se encuentra el complemento de Flash, de lo contrario envía un error.

Deben estar disponibles las librerías multicontacto de manera que sea capaz de reconocer eventos de este tipo para modificarlos a discreción.

Se necesita de una superficie multicontacto sobre la cual se pueda desplegar la

aplicación y sobre la cual se pueda trabajar, el tamaño y resolución del proyector no es de importancia. El rastreador debe ser capaz de detectar puntos de contacto, no es necesario que detecte objetos o presión sobre la pantalla.

4.3 Integración de elementos

En cuanto al software se refiere, se desarrolla la aplicación dentro del ambiente de desarrollo Qt. Esto es porque Qt posee ya una librería capaz de manejar el multicontacto, llamada Qtuio, basada en el protocolo TUIO.

Específicamente para el aspecto que se refiere al manejo de mapas y vista de calles, se utilizan Google Maps como el radar y Street View como el mapa, ambos corresponden a la capa de *widget*, los cuales se encargan del funcionamiento interno al mismo tiempo que la aplicación del usuario se encarga de mostrar el resultado obtenido [Klompmaker y Reimann, 2009], lo cual es manejado por la herramienta llamada QtWebKit, que se encarga de las peticiones y respuestas del servicio y traerlo a la aplicación, siendo así un servicio fuertemente acoplado. QtWebKit cuenta con un emulador de explorador de Internet capaz de leer cualquier código y desplegarlo en un componente llamado QGraphicsWebView; de esta manera QGraphicsWebView se encarga de manejar el código JavaScript y HTML que Google requiere para el manejo de sus mapas. Además de funcionar como un emulador de explorador Internet QGraphicsWebView tiene la particularidad de contar con características gráficas que permiten fácilmente la rotación del mismo, ya que hay que recordar que esto será desplegado en una mesa, por lo tanto es necesario mostrar las cosas en la pantalla al revés para que el segundo usuario las pueda ver correctamente.

El mapa específicamente es un elemento proporcionado por las librerías de Google perteneciente al servicio de Street View llamado “panorama”, mientras que el radar es un elemento predefinido por Google llamado “GMap2” el cual facilita el uso de marcadores y provee vistas aéreas que muestran la distribución de las calles alrededor de una ubicación.

Las zonas de control son rectángulos con una imagen de fondo, los cuales corresponden a la capa de interpretación.

El aspecto de la interfaz gráfica de usuario es manejado con un asistente de Qt llamado QtDesigner, el cual es capaz de manejar *widgets*, páginas web y componentes para facilitar la colocación de elementos en la pantalla. Los componentes creados en cuanto a interfaz y funcionamiento se refiere se analizan utilizando el simulador de superficies multicontacto previamente mencionado.

El desarrollo de las líneas de código que se refieren al aspecto de multicontacto es logrado a través del QtCreator en el lenguaje C++ y la librería multicontacto QTUIO la cual se puede descargar desde Internet y funciona como rastreador. (Figura 4.1).

En cada *widet* se agregan distintos elementos personalizados, lo que permite fácilmente la traslación, rotación y escala de cada uno de ellos, funcionando como botones en la aplicación. Todos estos componentes poseen la característica de recibir objetos multicontacto además de emitir señales y contar con ranuras, lo que se explica en la sección 4.4.

Desde el punto de vista de Qt, agregar características multicontacto a un componente requiere de agregar la librería al proyecto y a la clase además de sobreescribir el método encargado de la detección de eventos.

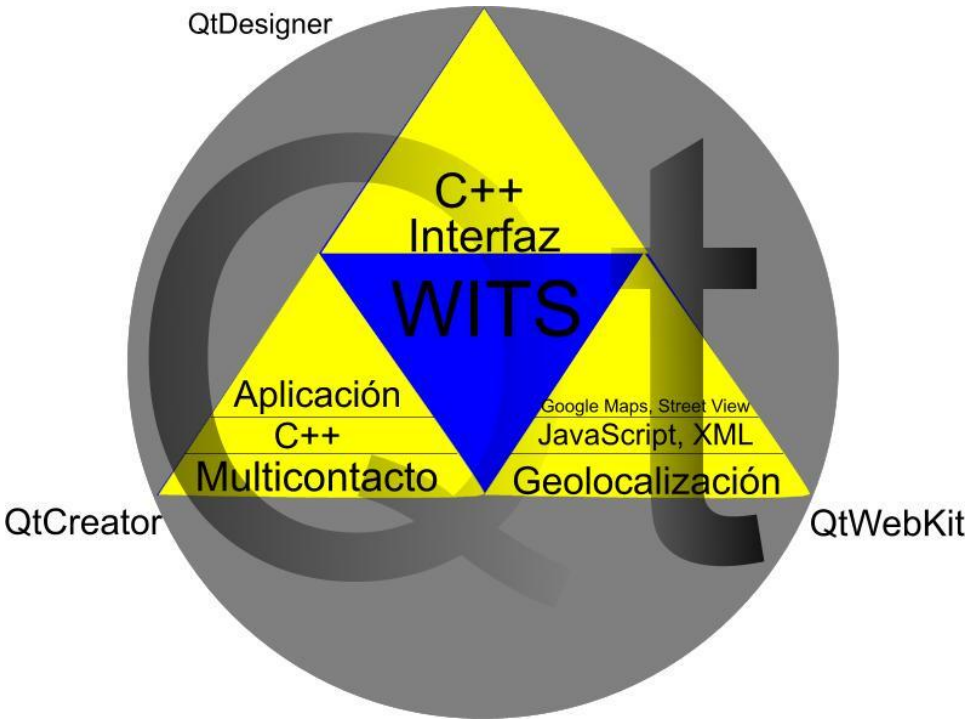

Figura 4.1-Diagrama de software.

4.4 Señales y ranuras

Se utiliza en gran manera una funcionalidad de Qt que es la de señales y ranuras. Esta permite avisar a otras instancias de objetos o incluso clases diferentes que algún objeto sufrió un cambio sin necesidad que la que cambió se percate del estado de los demás objetos. Se puede decir que es un estilo de “callback” con las mismas funciones, permitiendo trabajar con aplicaciones distribuidas de manera fácil dentro de una misma aplicación sin necesidad de implementar paradigmas distintos de programación como lo es el método de invocación remota.

En cuanto a la codificación se refiere, una ranura es un método cualquiera de un objeto con la implementación que el programador establezca con la única diferencia que se declara como ranura. Todos los métodos declarados como ranuras no deben regresar nada, es decir son de tipo “void”, lo que se compensa al poder enviar parámetros por medio de las ranuras.

Una señal puede ser recibida por varias ranuras simultáneamente y sin importar que esto se realice entre distintas clases ya que Qt se encarga de verificar que esto sea irrelevante.

La aplicación hace uso de estas específicamente para indicar el momento en que un usuario cambió de posición, la señal es recibida por el controlador quien envía una señal al mapa indicando que actualice su imagen. A su vez el mapa emitirá una señal cuando la nueva posición sea inválida y de esta forma regrese a la última posición. El controlador se encarga también de enviar una señal al mapa en el momento en que dos jugadores choquen para

4.5 Desarrollo del juego

El mapa y el radar, ambos pertenecientes a la capa de widget, son QGraphicsWebView (Figura 4.2) dentro de una escena, por lo tanto pueden ser rotados y se les puede inyectar código JSP, de esta forma se cambia gráficamente la posición en ambos. Para el caso del radar, primero se grafica la meta, después los jugadores y finalmente las zonas de tráfico. Internamente para el radar, cada uno de ellos representa un método distinto, ya que cada uno de ellos usa un color diferente. Esto se realiza por medio de identificadores, los cuales son números enteros asignados a cada radar, cuando el radar detecta que el siguiente jugador a graficar tiene el mismo número que el que le fue asignado lo despliega de color azul, indicando la posición de ese jugador.

Figura 4.2 Modo para un jugador.

La posición de cualquier ubicación se representa con un marcador y para diferenciar se usan distintos colores. Cada jugador verá en su radar un marcador de color azul que representa su ubicación en el mapa, los marcadores verdes representan a otro usuario, los rosas son zonas donde hay tráfico y por último el marcador rojo representa la meta.

La forma de avanzar se realiza presionando la zona de control con dos dedos (Figura 4.3) mientras que para dar vuelta hacia un lado se debe mover un dedo sobre la zona de control hacia el lado al que se desea mover (Figura 4.4).

Figura 4.3 Gesto para avanzar.

Figura 4.4 Gesto de giro a la derecha**.**

Originalmente se planteó que la forma de avanzar sería mediante la presión que se hacía sobre la superficie; es decir, se presionaba la palma de la mano contra la mesa y mientras más presión se hacía mayor la distancia que el usuario recorría. Sin embargo, el rastreador que se encuentra instalado no cuenta con estas características avanzadas de detección, por lo tanto se recurre a movimientos que reaccionan al número de puntos de contacto.

La capa de contenedor se representa por un componente de tipo QGraphicsScene, el cual representa un área de trabajo sobre la cual se pueden agregar cualquier cantidad de objetos, es decir cualquier cantidad de mapas, radares y zonas de control. Para el usuario esta escena se muestra como una ventana común e internamente para el sistema operativo representa de igual forma una sola ventana, eliminando así el problema del *focus*. En partidas con múltiples jugadores el contenedor es el encargado de establecer el punto inicial en el cual se encontrarán los jugadores, esto lo realiza mediante un llamado a WITS. Debido a la resolución del proyector utilizado todas las capas de contenedor tienen un tamaño de 1024 pixeles de ancho por 768 de alto.

La creación de los menús se realiza mediante ventanas con capacidad de recibir eventos multicontacto, una vez que se selecciona una se pasa a la siguiente. En la figura 4.5 se muestra el menú principal desde el cual se puede seleccionar partida para uno o dos

jugadores. No hay opción para más jugadores ya que el tamaño de la pantalla no es lo suficientemente grande para desplegar más elementos.

Figura 4.5 Menú principal.

En la figura 4.6 se muestra el menú de selección de ciudades, en el cual cada ciudad representa un nuevo rectángulo que es sensible al contacto humano y envía un parámetro de tipo cadena a la siguiente ventana, es decir el contenedor, indicando la ciudad elegida. En seguida el contenedor sitúa a los jugadores en la ciudad seleccionada.

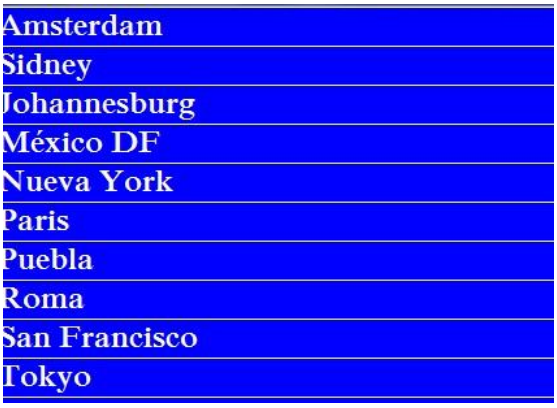

Figura 4.6 Menú de selección de ciudad.

La figura 4.7 muestra la versión de juego para dos jugadores, cada jugador cuenta con su respectiva zona de control, mapa y radar. La zona de control es el rectángulo azul con el

símbolo de una mano, el mapa es el elemento de Street View. El radar se muestra con los diferentes marcadores indicando zonas de tráfico, meta y contrincante. Algunos componentes se encuentran al revés ya que esta imagen se estará proyectando sobre una mesa, habiendo un jugador en cada lado.

Figura 4.7 Versión de juego para dos jugadores.

4.6 Resumen del capítulo

En este capítulo se describe los componentes que son utilizados en WITS para poder cumplir con los objetivos propuestos. Se indica la forma en la que cada componente se entrelaza con los demás y se describe el mecanismo de señales y ranuras que permite la comunicación entre los componentes funcionando como un sistema distribuido.

En el siguiente capítulo se analizará qué tan eficiente resulta esta implementación, en base a la opinión de usuarios que probaron el sistema. Por último se dará una conclusión de la eficiencia de esta implementación además de su trabajo a futuro.