PCYP streets
Manual
tecnico
Saul Palestina Cruz.

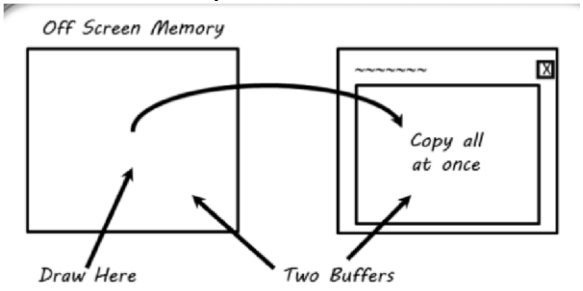
Mireya Tovar Vidal.

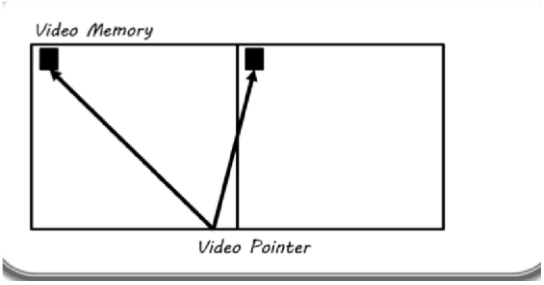
La clase Simulacion.

Estaclase debe forzosamente heredar directamente de JFrame e implementar Runnable, esto se debe a una gran variedad de razones, las cuales cubriremos de la forma mas breve posible en las siguientes secciones.

Active rendering.

El renderizado haciendo uso de la biblioteca awt de java suele requerir frequentes a la funcion repaint(), un medio para resolver esta ineficiencia (considerando que el renderizado en programas que utilizan graficos deben ser inmediatos) es el uso de el active rendering (o renderizado activo) el cualhace uso de un buffer doble, realizando los calculos en ambos y mostrando en pantalla unicamente uno.





Este renderizado permite hacer un uso mas adecuado del procesador, ademas de ser el concepto principal de el siguiente punto.

Las operaciones requeridas son dadas mediante llamadas constantes a un metodo que llamaremos cicloDeRenderizado(), el cual haciendo uso de dos ciclos obtiene el atributo Graphics de nuestro respectivo marco (o algun tercero como por ejemplo Canvas, visto a continuacion), con el cual podemos comenzar a trabajar de la misma forma tradicional en la que se usan las clases y bibliotecas de awt. Finalizado por vez primera el renderizado, se asegura que nuestro buffer de graficos se basee con dispose(). El primer ciclo comprueba si nuestro buffer tiene cambios (se ha dibujado un nuevo atributo), igualmente comprueba si el primer renderizado esta listo para mostrarse con show(), despues el ciclo continua o se pierde con contentsLost().

Uso de un hilo para el programa principal de renderizado.

El uso inapropiado de awt suele provocar ciertos efectos indeseables en la escena, por lo cual es recomendable hacer uso de un hilo independiente de nuestro proceso pesado main. Por esta razon ejecutarAplicacionDeRenderizado() hace uso de el evento invokeLater de SwingUtilities para estar seguros de que el hilo no ha empezado a ejecutarse hasta que el programa haya terminado de iniciar, el tiempo que tarda varia segun el equipo y su compatibilidad con java o el sistema operativo que se este utilizando. Para cerrar el programa se crea un nuevo evento, windowClosing(), el cual ejecuta nuestro metodo ventanaAlCerrar(), el cual detiene el hilo de renderizado, esto es perfecto para evitar que el programa se cierre hasta que se hayan guardado cambios importantes,como por ejemplo registros o informacion que ha quedado congelada por el cierre inesperado o inapropiado.

Ahora bien, los atributos de esta clase estan divididos en atributos de renderizado (fuente de texto de la ventana, el ancho y largo de la ventana, el color de fondo, etc) y los atributos que componen la escena (direcciones y semaforos). El constructor no es recomendable de usar, puesto que necesitamos constantemente metodos de esta misma clase, por este motivo usamos inicializar(). Si bien este metodo no es el primeor en ejecutarse en el programa, si es el que inicializa los atributos de la escena, en este caso las coordenadas reales de nuestro mapa, ademas de el transito.

Ciertamente el primer metodo es crearYMostrarGUI(), el cual inicializa todos los atributos de nuestra ventana, ademas de crear el hilo ya mencionado anteriormente (el

hilo que se encargar de renderizar la escena). Dentro de este metodo definimos la causa que provoca el evento componentResized() el cual es el hecho de que el usuario amplie o disminuya el tamanio de la ventana (este evento hace que de lugar uno de los aspectos mas importantes de este programa, mencionadoa continuacion).

Dentro del metodo run() tenemos el ciclo que mantiene al hilo en ejecucion funcionando hasta que se de de el evento de cierre (llamando a ventanaAlCerrar()), ademas de las constantes invocaciones a el cicloDinamicoDeDibujado(), el cual realiza otro llamado a el ya mencionado metodo renderizadoDelMarco(), el cual a su vez realiza las operaciones explicadas con brevedad anteriormente, aunque quiza el punto mas remarcable es que el realiza los llamados a el renderizado principal, o renderizar().

A grandes razgos este es el procedimiento unicamente para el marco, ahora la forma de redimenzionar los tamanios de la escena y ajustar apropiadamente las dimensiones de nuestra ventana se explican a continuacion.

El metodo obtenerTransformacionViewport() y la clase Matrix3x3f.

Para poder realizar las conversiones debemos hacer uso de una matriz tridimensional a la cual podemos aplicarle las transformaciones y escalamientos necesarios para el dibujado apropiado, ver metodos de la clase Matrix3x3f. En cuanto a elprocedimiento, primero obtenemos las coordenadas de la ventana y realizamos una division por las coordenadas de nuestro mundo, el cual tiene una escala desde -2 a 2 en ambas dimensiones. El metodo devuelve esta matriz para poder aplicarla en todos nuestros puntos de dibujado, y asi escalarlos con respecto a la escena y la ventana.

El funcionamiento del programa.

Hay 4 tipos de clases que componen la logica principal del programa:

- De utilidad, una de ellas es el recurso compartido y otras son las clases de graficos creadas manualmente.

- De ejecucion, encargadas de producir y consumir recursos de las clases anteriores.
- De representacion, las cuales componen a el recurso compartido para cumplir apropiadamente la representacion de los semaforos y los automoviles.
- De implementacion, el cual se encarga de redimensionar el marco y la escena.

Clases de utilidad.

Estas clases son utilizadas por las clases de ejecucion e implementacion para realizar los calculos necesarios y administrar los recursos. Centrandonos en esta ultima nos damos cuenta de que en la logica delprograma intervienen varias direcciones, a las cuales les corresponde uno y unicamente un semaforo, una coleccion de vectores que definen su direccion y una raiz que representan la lista ligada de vehiculos que siguen esta direccion. De esta forma, la clase Direccion.java se encarga de que ninguna clase de tipo ejecucion o de implementacion adquiera uso desordenado de los recursos.

Clases de ejecucion.

Encargadas de producir y consumir los recursos de las clases de utilidad y representacion, solicitan continuamente acceso a su determinada direccion para efectuar operaciones eficientes y rapidas en su respectivo monitor, evitando que existan problemas de sincronizacion.

Clases de representacion, ubican con dimensiones a escala a los objetos de nuestra escena.

Clases de implementacion, hacen uso de la clases de utilidad para realizar sus respectivas operaciones, ademas de requerir acceso exclusivo a su monitor para segurarse de no dibujar nada que no este en este recurso compartido.

Clases de ejecucion, un vistazo mas cercano.

En las clases de ejecucion contamos con entidades "inteligentes" que dependen totalmente de su monitor para realizar operaciones en sus atributos. La clase transito se asegura de que ciertos vehiculos entren en la escena de forma ordenada y aleatoria, esta se encarga de generar conductores para que manejen el vehiculo. Los coductores por su parte tienen que tomar en cuenta el vehiculo que se encuentra enfrente suyo y la direccion que debe seguir su vehiculo, ademas del estado de su semaforo. Y por ultimo el interruptor contiene todos los semaforos de neustra escena, de esta forma la informacion no se pierde (coordenadas y estado) para que los conductores tengan tiempo de pasar cuando sea su turno, en todo el programa contamos con multiples conductores, pero unicamente un interruptor y varios hilos de transito limitados

Cabe mencionar el hecho de que todos los hilos conductores acceden mayormente al desplazamiento de vehiculos de la clase Direccion, el cual realiza todas las oepraciones que manejan el movimiento, el movimiento es rectilineo usando el algoritmo de brensenham para lineas rectas.