Colision Simulator

Proyecto #1

Taller de Programación

Estudiante:

Carlos Andrés Mata Calderón

Carne:

2019033834

Profesor:

Jeff Smith

I Semestre, 2020

Contenidos

Tabla de ilustraciones

Imágenes

Tablas

Introducción

Sin lugar a duda, los accidentes automovilísticos son la principal causa de muerte a nivel mundial, según los datos confirmados por la OMS. En búsqueda de erradicar esa elevada estadística, que afecta al mundo entero, es que la tecnología intenta dar una posible solución, que aunque no elimina por completo las muertes de este tipo si posibilita la oportunidad de crear un sistema especial del vehículo con el fin de detectar la cercanía de otros vehículos con los que podría colisionar, dando una alerta inmediata y efectiva al conductor evitando el choque, además, si el conductor por alguna razón no logra reaccionar a esta primer alerta, el sistema de manera automática realiza un freno de emergencia total.

En la empresa Volvo se creó el Colision Warning Break Emergency (CW-EB), que tiene la funcionalidad anteriormente explicada. Sin embargo, estos quieren agregar una nueva función, la cual consiste en si el auto se encuentra a otro vehículo cerca, este realizará un cambio de carril, para evitar un posible choque.

El siguiente proyecto diseña un simulador que implementará esta nueva función con el fin evitar una colisión frontal o lateral.

Descripción del Problema

Con la idea de brindar los servicios como Ingeniero en Computadores se crea un simulador que ejemplifique como se comportaría el vehículo en la vida real. Dentro del sistema se evidenciará los eventos que pondrán a prueba la reacción del sistema.

Todos estos tipos de eventos se tomarán en cuenta para la realización del proyecto, con el fin de mostrar una experiencia igual o similar a la nueva funcionalidad del CW-EB, además se agregarán las funcionalidades de frenado y choque, que respectivamente simulan el viejo CW-EB y que pasaría con un vehículo sin CW-EB

Los eventos se clasifican en eventos de Esquivar, Frenado y Choque:

Eventos de Esquivar

¿Qué pasaría si hay un auto cerca dentro del mismo carril que se conduce?

¿Qué pasaría si hay un auto al lado y se intenta pasar a ese carril?

¿Qué pasaría si hay dos autos muy próximos, que no dan el campo suficiente para cambiar de carril?

Eventos de Frenado

¿Qué pasaría si hay un auto cerca y en el mismo carril que se conduce?

¿Qué pasaría si hay un auto al lado y se intenta pasar a ese carril?

Eventos de Choque

¿Qué pasaría si esta desactivado el CW-EB y se acerca mucho un auto?

Por último, se agregará un salvapantallas de duración configurable al usuario que mostrará, el método de animación que se implementó en la simulación del CW-EB.

Metodología

Se utilizará el lenguaje Python, debido a la extensión de librerías y simplicidad de sintaxis del lenguaje, además de contar con un gran apoyo de la comunidad. Utilizará el paradigma de programación llamado como POO (programación orientada a objetos), como el principal método para realizar el proyecto.

Las librerías por utilizar son:

- Tkinter, se utilizará como la interfaz principal la cual mostrará de forma amigable con el usuario la ejecución del programa.
- Random, se utilizará como el método que genera eventos aleatorios del programa. Ejemplo, la aparición aleatoria de autos, movimientos de las bolitas, colores y imágenes del programa.
- Time, se utilizará por conveniencia del programador que, para retrasar algunos eventos del programa.
- Thread, se utiliza como auxiliar, para realizar las animaciones del programa, se tiene que importar de la librearía threading la clase Thread.
- Sys, se importará al sistema operativo debido que se necesita quitar el límite recursión, esto es para la prevención de errores relacionados con límite de ejecución de funciones. Ejemplo Python por defeult tiene un límite de llamadas a una función entre 950 a 1100.
- Messagebox, se importa de la librería Tkinter debido a la manera que se encuentra programado la Tkinter se tiene que llamar a Messagebox, de esta manera.
- Winsound, es una librería propia de Python Windows Version, que ayuda en importar sonidos en la carpeta donde se encuentra el programa.

Por último, se utilizará la ayuda del profesor del Curso Taller de Programación de Computadores Jeff Smith, su asistente Nicolás Vigot Orozco, los tutores de Programación del I Semestre 2020 y amigos que ya pasaron el curso Taller de Programación, además de utilizar como fuente principal de consulta el Internet, con el fin de aclarar las dudas relacionadas con el proyecto.

Análisis de Resultados

Ejemplos de corridas del código

Este proyecto consta de múltiples funcionalidades, las cuales al final logran su cometido de emular correctamente el sistema de frenado.

Este simulador cuenta con dos ventanas de diversas características. La ventana del menú y la ventana de la simulación. Todas las ventanas utilizadas en esta simulación se manejan con clases, más adelante se explicará lo condujo a manejarlo a programar de esta forma.

Cuando se inicie el programa Simulacion_Colision.py o Simulacion_Colision_No_Sound.py aparecerá lo siguiente:

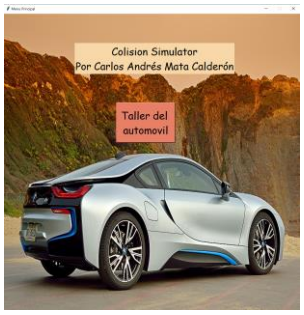

Ilustración 1:Ventana de Menú

En esta ventana solo se encuentran un par de labels (imagen, texto) y el botón (Taller del automóvil). Este botón se encarga de la modificación de la ventana menú por primera vez.

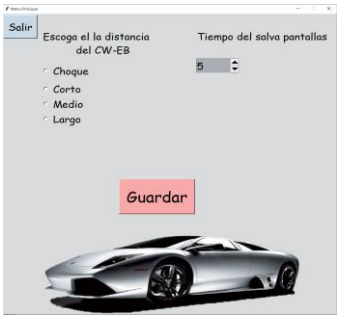

Ilustración 2:Taller de automóvil

Luego de presionar el botón (Taller del automóvil) aparece esta modificación de la ventana menú que se encarga poner los parámetros iniciales del automóvil que son obligatorios. Estos parámetros son si el CW-EB estará activado o desactivado, la distancia del parámetro de proximidad, y también la duración del salvapantallas. Cuando se presiona el botón (Guardar), sonará un sonido de reparaciones y se hundirá con una espera 5 segundos, para seguir la ejecución del programa.

Ilustración 3:Salva Pantallas

Después de salir del taller, se vuelve a modificar la ventana menú para muestre la animación de unas bolas de colores rebotando y cuando estas bolas chocan con el borde de la ventana realiza un sonido rebotar (‘boing’). La duración de esta animación dependerá de la cantidad seleccionada de segundos en el Taller del automóvil.

Ilustración 4: Pantalla Start

Por último, la ventana se modifica una vez más, apenas que se termina de modificar la ventana se escuchará un sonido, y cuando se presione el botón se reproducirá un sonido, y después de 2 segundas dará inicio a la ejecución de la simulación.

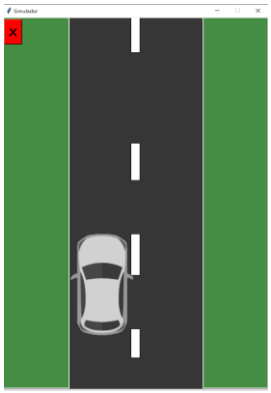

Ilustración 5: Ventana de la Simulación

En la ventana de la simulación, se muestra la animación de esquivar, frenar o colisionar, todo depende del modo que se haya seleccionado, en el Taller del Automóvil. El automóvil de color plateado será el vehículo que en conducir.

A continuación, se dará respuesta a las preguntas de los eventos de Esquivar.

Eventos de Esquivar

Este evento cambia de carril cuando el automóvil se encuentra muy cerca otro, además bloquea el movimiento del conductor si este trata de acercase al otro vehículo.

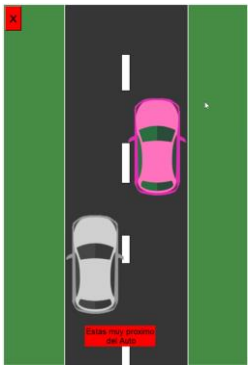

Ilustración 5:Evento Esquivar 1

Aquí se muestra como el auto se encentraba cerca otro vehículo, lo que producía muestre un mensaje de advertencia.

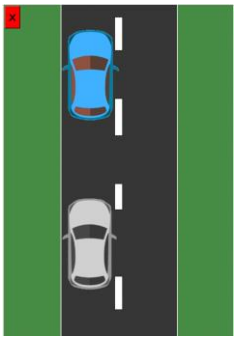

Ilustración 6: Evento Esquivar 2

En esta ilustración se ve como el auto se comporta, antes de realizar el movimiento de cambio de carril.

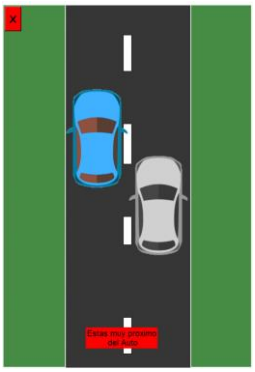

Ilustración 7: Evento Esquivar 2.1

En la ilustración 7 muestra como el auto cambió de carril de forma automática.

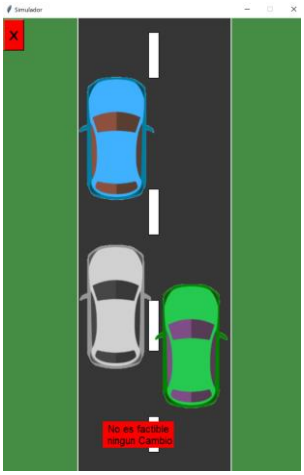

Ilustración 8: Evento Esquivar 3

Aquí se da el ejemplo qué sucedería, si no es posible realizar ningún cambio de carril.

El programa, inmediatamente dará un mensaje que se acabó la simulación, ya que tuvo que frenar el auto, debido al cambio no factible. Después de que se muestre el mensaje de advertencia, se ejecutará un sonido de claxon.

Eventos de Frenado

Los eventos de Frenado son los encargados de decir si un auto se encuentra muy cerca del otro y frena si el usuario no realiza ninguna accción.

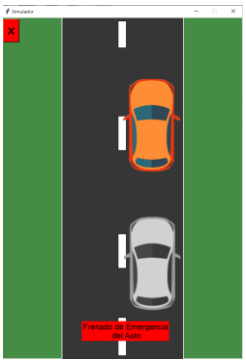

Ilustración 9:Evento Frenado 1

Cuando el auto se encuentra al frente del otro vehículo y el parámetro es mayor a la distancia entre autos, se realizará el frenado de emergencia y seguido de un sonido de claxon. Además, que mostrará un mensaje que explica lo que sucedió.

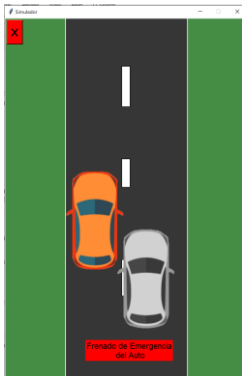

Ilustración 10:Evento Frenado 2

En la imagen anterior que sucede si el conductor intenta de acercase a los laterales del otro automóvil. Este evento es simular al evento Esquivar 2 cuando el conductor intenta cambiarse de carril, ya que el CW-EB funciona y provocará que se frene el auto a una distancia segura para evitar el choque.

Eventos Choque

Este es el último evento del simulador, este simula que sucedería si se desactiva el CW-EB.

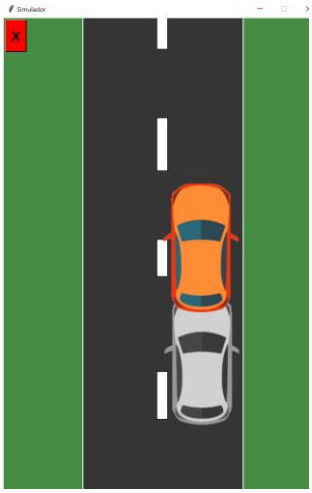

Ilustración 11: Evento Choque

Cuando el vehículo choca con otro auto, se escucha un sonido. Y el auto se detiene debido a que chocó.

Para regresar a la ventana menú, se implementó un botón que realiza la acción de cerrar la simulación y ejecutar la ventana menu

Explicación de la Lógica del programa

Como se explicó anteriormente, se utilizó el paradigma de programación POO, debido a la facilidad de utilizar la librería Tkinter de esta forma, por la razón que, si se necesita utilizar un obejto de Tkinter en otro lugar de la clase simplemente se ponía como un atributo de la ventana.

Además, si se ocupaba crear o destruir una ventana simplemente se llamaba al atributo ventana de la clase.

Se crearon un total 5 clases en todo el programa. A continuación se explicará los atributos y métodos de dichas clases.

- Clase Menu:

Class Menu()
- self.menu -self.fondo_label - self.nombre_label - self.boton_configuracion - self.fondo_label_3 - self.label_info_1 - self.label_info_2 - self.boton_guardar - self.boton_salir - self.radio_buttonx1 - self.radio_buttonx2 - self.radio_buttonx3 - self.radio_buttonx4 - self.spinbox_mode - self.spinbox_tiempo - self.canvas_animacion - self.canva_fondo - self.fondo_label_4 - self.boton_start
Método init Método ingresar Método ingresar2 Método choice Método ingresar3 Método animación Método create_circle Método ingresar4

Tabla 1: Diagrama de clase de Menú

El diagrama de clase anterior muestra todos los atributos y métodos que conforman esta clase.

Los atributos de esta clase son números por la razón, que es una ventana que debe estar modificándose así misma con los diferentes métodos implementados. El motivo por el cual se hace esto, fue por los contratiempos que generó el uso de clases con ventanas de Tkinter, más adelante se dará énfasis esto último.

- ✓ Método init: en el método init se crearon los diversos atributos, del modo de menú, taller de automóvil, salvapantallas e inicio de simulación. Con sus respectivos labels, canvas, botones, radio_botons, spinbox y variables que se necesitarán para configurar la ventana. Los únicos atributos que tienen posición por defult son: fondo_label, nombre_label, boton_configuracion.

- ✓ Método ingresar: se encarga de reproducir un sonido, destruir la ventana e iniciar la ventana de simulación. Este método se ejecuta cuando se presiona el botón de start, del modo inicio de simulación.

- ✓ Método ingresar2: pone de color la ventana diferente, además de esconder todas las configuraciones del modo menu y pone la posición del modo Taller de automóvil.

- ✓ Método choice: este encargará de toda la lógica detrás del guardado de las configuraciones. Para acceder a esta función se debe presionar el botón de Guardar, que al presionarlo ejecutará un sonido.

- ✓ Método ingresar3: se inicia al presionar el botón de salida del modo taller del automóvil, este no funcionará si previamente no se ha guardado la configuración del auto. Además, se encarga de cerrar los archivos, quitar el modo taller de automóvil y poner el modo de salvapantallas.

- ✓ Método animación: se ejecuta al final del método ingresar3, recibe el tiempo digitado por el usuario y se encarga de llamar a la función create_circle, está también se encargará medir el tiempo de la animación en la pantalla. Cuando termine el tiempo se llamará al método ingresar4.

- ✓ Método create_circle: este método recibe el tiempo, y se encargará de llamar a la clase Bolitas, con el de crearlas. También se crea un hilo que da el movimiento a la bolita, por último tiene condiciones sobre la duración de la animación, esto es para que, no genere errores si el usuario digita un tiempo muy corto de animación.

- ✓ Método ingresar4: este método es el que se encarga de borrar finalizar la animación y borrarla de la pantalla, para poner el modo inicio de la ventana menú, y también ejecuta sonido cuando se le llama.

- Clase Bolitas:

Class Bolitas()
- self.menu -self.canvas -self.xspeed - self.yspeed - self.bola - self.terminar
Método init Método move

Tabla 2: Diagrama de clase de Bolitas

El diagrama de clase anterior muestra todos los atributos y métodos que conforman esta clase.

Esta clase recibe el canvas donde se pondrá las bolitas.

Los atributos de esta clase son principalmente para la creación aleatoria del tamaño y color de la bolita.

Los métodos de esta clase son pocos ya que, el movimiento de la bolita es muy simple.

- ✓ Método init: aquí se encuentra como se verá la bolita cuando se llama a la clase, sus colores son aleatorios, además de su radio (multi), y finalmente se crea la bolita.

- ✓ Método move: se encarga del movimiento de la bolita, con los atributos self.speed se le dará la velocidad con la que se mueve, además que al final está el análisis si la bolita está a punto de chocar con un borde del canvas, para que rebote y ejecute un sonido de ‘boing’.

- Clase Simulación:

Class Simulacion()
- self.ventana - self.canvas_calle -self.yspeed -self.raya1 -self.raya3 -self.raya5 -self.raya7 - self.warning_label - self.carro -self.car_izquerda -self.car_derecha -self.loops_thread
Método init Método decoración Método presion_tecla_hilo_start Método move_ramdoms_cars Método cw_eb Método choque Método loop Método salir

Tabla 3: Diagrama de clase de Simulación

El diagrama de clase anterior muestra todos los atributos y métodos que conforman esta clase.

Se quiere señalar que al puro principio se creó con el fin utilizarla como un auxiliar para terminar todas las animaciones de la Simulación, su nombre es 'terminar', en ese momento almacena un valor de verdad. Cabe agregar que esta bandera se encuentra en casi todos los métodos de relacionados con la animación de la simulación, esto anteriormente también se utilizó en la animación de las bolistas rebotando, pero con un nombre distinto de variable.

Los métodos de esta clase son unos de los más complejos del programa, ya que tiene muchos métodos que se encargan de la lógica del CW-EB, de animación o auxilares, estos últimos son principalmente los relacionados con los Threads de la ventana.

- ✓ Método init: como explico aquí principalmente se encuentra objetos de Tkinter, sin embargo, se quiere resaltar que aquí también se encuentra la creación instancias de los objetos Carrito (carro principal) y Carros Aleatorios (son los carros que se esquivarán),casi al final se encuentra la llamada a un método auxiliar de hilos, y antes del mainloop() de la ventana que crea un hilo para el método loop.

- ✓ Método decoración: este encargará de generar la ilusión de movimiento del carro atraves de una autopista, aquí se mueve unos rectángulos de color blanco que están en el canvas_calle. Su movimiento es muy similar al de las bolitas.

- ✓ Método presion_tecla_hilo_start: se encarga de crear un hilo para el método del carro principal llamado self.carro.presion_tecla_aux.

- ✓ Método move_ramdoms_cars: se encarga de generar los hilos de la animación de decoración, y de los métodos de movimientos de los carros aleatorios del programa (self,'carro aleatorio'.move).

- ✓ Método cw_eb: este método que se encarga de recibir la proximidad que se escogió en la ventana menú, además de contar con la funcionalidad principal retornar una lista con un valor booleano y un string, además se en encarga de verificar si el auto principal se encuentra cerca de otro en Y.

  Cabe destacar, que si no se encuentra cerca ningún auto retorna un valor Falso, de lo contrario será un valor Verdadero junto a la identificación del auto.

- ✓ Método choque: es muy parecida a la anterior, pero con la diferencia que no recibe un parámetro si no que ya tiene uno, que es el mínimo del tamaño del auto.

- ✓ Método loop: es el método más extenso del programa, este primero recibe cuál de las tres opciones se escogió, esto es a través del archivo que guardó los parámetros de configuración, después entra a un bucle con ayuda de la bandera, que ejecuta los métodos de comparación de todas las circunstancias de cada evento.

  El primer evento que se puede ejecutar es el de esquivar, solo entra este evento si el usuario digitó que quería esta opción. Este llama al método de cw-eb, que, dependiendo del valor que retorne, llamará al método **esquivar que trae incorporado el auto.** Por último, si el cw-eb dice que, si los dos autos están muy cerca, se frena el carro, con el fin de prevenir el choque.

El segundo evento es el de frenado, realiza a primera instancia lo mismo que esquivar con las dos diferencias , que este no solo depende del método cw-eb sino también del método de **frenado que trae incorporado el auto**, y la otra diferencia es que este frena cuando el parámetro detecta que la posición del auto respecto al otro es menor, con el fin de prevenir el choque.

El tercer y último evento utiliza los métodos de choque como la referencia de la posición entre los autos, además depender otro método **choque que trae incorporado el auto**, con el fin de dar la impresión de que el vehículo chocó.

Todos estos eventos de frenados o choque, hace que la bandera adquiera un valor de Falso, lo cual hace terminar la animación y muestra en pantalla un mensaje del porqué se frenó o chocó el auto.

- ✓ Método salir: se ejecuta cuando el usuario le da clic al botón con la X, este se sale de la ventana simulación, y ejecuta la clase Menu, para volver a cargar su ventana.

- Clase Carros_Aleatorios:

Class Simulacion()
- self.canvas - self.Obstaculo_X self.Obstaculo_Y self.yspeed -self.lista -self.obstaculo
Método init Método move Métodoget_Obstaculo_X Métodoget_Obstaculo_Y Métodoget_Obstaculo_y_Arriba

Tabla 4: Diagrama de clase de Carros_Aleatorios

El diagrama de clase anterior muestra todos los atributos y métodos que conforman esta clase.

Esta clase recibe el canvas donde se pondrá el auto y una lista imágenes. Además, que aquí se utiliza la bandera creada en la clase Simulacion, esta se utiliza para terminar el movimiento del auto.

En esta clase gran parte de sus atributos y métodos son similares a la clase Bolita, ya que se utiliza para darle apariencia y movilidad al auto.

- ✓ Método init: se encarga de darle apariencia aleatoria del auto, además una posición aleatoria del mismo. Se crea un atributo canvas para movilizarse en él y crear el auto dentro de él.
- ✓ Método move: este método se encarga del movimiento aleatorio en y cuando llega al final del canvas, además de cambiar el color del auto para que de la ilusión de que es un auto distinto la próxima vez que sea logre ver.
- ✓ Método get Obstaculo X, get Obstaculo Y y get Obstaculo y Arriba: son métodos auxiliares que se utilizan en los métodos de comparación de la posición con el auto principal.

- Clase Carrito:

Class Simulacion()
- self.canvas - self.Obstaculo_X self.Obstaculo_Y self.yspeed -self.lista -self.obstaculo
Método init Método move Método presion_tecla_aux Método get_Car_X Método get_Car_Y Método esquivar Método frenado_automatico Método choque_de_carro

Tabla 5: Diagrama de clase de Carrito

El diagrama de clase anterior muestra todos los atributos y métodos que conforman esta clase.

Esta clase recibe el canvas donde se pondrá el auto y una lista imágenes. Además, que aquí se utiliza la bandera creada en la clase Simulacion, esta se utiliza para terminar el movimiento del auto.

- ✓ Método init: se encarga de poner las posiciones iniciales del auto, que siempre son las mismas, además de escoger su respectivamente imagen de la lista de autos.

- ✓ Método presion_tecla_aux: se encarga de recibir cual tecla digito el usuario para mover el auto, ejemplo <Flecha Derecha> el auto se mueve a la derecha, etc.
- ✓ Método get Car X, get Car Y son métodos auxiliares que se utilizan en los métodos de comparación de la posición de los otros autos.
- ✓ Método esquivar: este método se encarga del movimiento en X del auto si este se encuentra en el mismo carril que él. Se mueve hasta que el auto no se encuentre en el mismo carril.
- ✓ Método frenado y Método choque: estos son muy parecidos ya que realizan la misma lógica del método esquivar con la diferencia que retorna únicamente valores booleanos. Hay que destacar que los parámetros de proximidad de frenado son mucho más grandes que los de choque.

Hay una función externa que se creó aparte de las clases, se llama sonidos, como lo dice en su nombre se encarga de reproducir sonidos en hilos para que no afecte al funcionamiento del programa.

Al final del programa, se ejecuta la clase Menu, para dar inicio cuando el archivo py se abra.

Se recomienda hacer un respaldo del código, y leerlo, para un mayor entendimiento de la lógica de este mismo.

Problemas a lo largo del desarrollo del proyecto

Los Threads con ventanas de Tkinter y el uso de clases como método de creación de las ventanas de Tkinter con incompatibilidad al Importar imágenes dentro de él: Al principio se desconocía la incompatibilidad de los Threads con ventanas en Tkinter. La ventana del splash animado no podría abrir ya que al principio era una ventana extra del programa en una clase llamada Animacion. La clase Animacion al final de su método init ejecutaba un método que metía en la llamada a la clase Simulacion en un hilo, esto era porque se tenía hacer un time.sleep, lo cual provocaba errores en la ejecución del programa.

Además de una incompatibilidad de abrir 3 ventanas con el paradigma de programación POO al importar imágenes dentro de dichas ventanas. Este fue el mayor problema durante el desarrollo del programa, ya que se desconocía de esa incompatibilidad, se estuvo alrededor de 3 días averiguando él porqué de este problema.

Debido que el proyecto se encontraba en una fase ya muy avanzada, para modificar todo el proyecto. Al final con ayuda de un tutor Santiago se entendió por qué del error.

Este se Resume a que solo podría abrir dos ventanas e importar imágenes con el paradigma de programación POO. Por lo que se tuvo que modificar la pantalla de menú para que hiciera todas las funciones que necesitaba el proyecto.

Salida repentina del usuario: Si el usuario se sale antes de que la simulación encuentre una situación de frenado de emergencia o choque. Python tirará un error que se terminó el while antes tiempo, por lo que tira un error en la consola. Se trató de solucionar con la bandera terminar, poniéndola antes de la ejecución de Threads y cambiando el valor de verdad en el método salir de la simulación, sin embargo, no se logró solventarlo.

Sonidos del juego: Los sonidos del juego generan ciertos errores superficiales a la ejecución del programa ejemplo, si se está reproduciendo un sonido, este no dejará que otro sonido sea reproducido por lo que se trató de solventar el problema mediante el uso de la librería time para que el usuario de un tiempo a que terminar el sonido que se está reproduciendo. Ejemplo, se hunde botón de Start y Guardar.

Además, que, si se ejecuta con Mac o Linux, el programa no funcionará por lo que se creó una versión especial para estos, pero sin sonido, porque la librería Windsound es exclusiva de Windows Python Version. Este segundo archivo se llama Colision_Simulator_No_Sound.py.

Uso del tiempo: Se tardó mucho tiempo, dar en solución a los problemas que presentaba el código cuando se creaba un nuevo método en una clase. En especial el primero problema anteriormente explicado, por lo que varias funciones como el movimiento de árboles pasando por el pasto de color verde , el cambio de apariencia del auto cuando este chocaba, un mejor acomodo del código y borrar ciertos comentarios que están demás en el código, mejorar documentación, no se pudieron realizar.

Documentación del proyecto: La documentación del proyecto, se prevé que tenga varios errores ortográficos y de redacción, debido a la inexperiencia en la documentación de este tipo proyectos, además de un largo tiempo de la última vez que se tuvo que redactar un proyecto de tal magnitud.

Bitácora

Sábado 30 de mayo:

Ese día solo se dio la tarea de entender el proyecto, además de pensar cómo se iba a programar el proyecto, y se sugirió la utilización de programación orientada a objetos (POO), debido la interacción entre ventanas y clases de Tkinter.

Lunes 1 de junio:

Este día se terminó de decidir que si se utilizara el paradigma de programación POO, para la creación de ventanas. Además de iniciar la programación del proyecto con la creación entre ventanas y los widgets de Tk.

No obstante, no se logró una buena interacción entre las ventanas.

Además, esa misma noche, para no seguir no el retraso de las ventanas, se empezó a programar, en la ventana de Simulación. Ese se día se logró mover caja negra que representaba el auto principal, de derecha a izquierda, arriba hacia abajo, etc. Además de lograr que el auto no traspasará la ventana cuando llegaba al límite de esta.

Martes 2 de junio:

Se acudió a las Tutorías con Daniel Fallas a las 4 pm, que ayudó un poco con la iteración entre ventanas, sin embargo, solo se logró hacer que funcionaran 2 de las 3 ventanas. Él también te dio varios tips del acomodo del código, ya que la clase de Carrito inicialmente estaba dentro de la clase Simulacion, lo que provocaba que se viera muy desordenado, con su ayuda se cambió esto.

Por último, en la noche se crearon dos cuadritos blancos sin movimiento, que representarían los otros autos.

Miércoles 3 de junio:

Se programó el movimiento, de los cuadrados blancos, y se logró que su movimiento y velocidad fueran aleatorios.

Jueves 4 de junio:

Se acudió nuevamente a tutorías con Daniel, con el fin de que te ayudará a pensar en la lógica del programa respecto al CW-EB. Ese día se hicieron grandes avances, debido a la ayuda de Daniel,terminada la tutoría se tendría una idea más clara de lo que se tenía que hacer esa noche.

Primero se programó el método cw-eb, luego el método esquivar, además de mejorar el movimiento de los autos aleatorios y también de crear las funciones que generaban los hilos de los movimientos del programa. Y, por último, una versión preliminar del método loop.

Viernes 5 de junio:

Solo se investigó como se podría terminar la Simulación ante los eventos, de frenado emergencia o choque. Y se encontró que una 'bandera' este podría terminar.

Sábado 6 de junio:

Con el conocimiento adquirido de como parar la animación se crearon todos los eventos de loop, para que este logrará terminarse, además de crear la lógica de frenado y choque, estos dos estarían basados en la lógica de esquivar y cw-eb. Solo que con parámetros distintos y acciones.

Además de crear, la salida de animación hacia la ventana principal, pero este último no se pudo ya que tiraba unos errores de incompatibilidad de los Threads con Tk, al meterlos dentro Threads. En ese momento se desconocía de su incompatibilidad.

Domingo 7 de junio:

El domingo en su gran mayoría el código tendría casi listo toda la simulación, por lo que se decidió darle imagen por fin a todos los autos. Sin embargo, se descubrió, que, al abrir la ventana desde el menú del programa. Este no funcionaba

Ese día se le consulto a varias personas, como Nicolas, amigos de computadores y computación, además de una larga búsqueda por Internet como solventar el problema. No obstante, se encontraba el porqué de los errores del programa.

Un mío amigo me enseñó un código realizado por él con varias ventanas, y me comentó que el no destruía las ventas cuando se salía de estas sino solamente se escondía. Pero este método no funcionó ya que igualmente generaba errores.

Sin embargo, me dijo que una ventana de su código no se escondía, sino que se modificaba constantemente, por lo que mi amigo me dio la idea de realizarlo así, aunque tenía que cambiar gran parte del código. Por lo que ese día no se adoptó por ninguna de esas posibles soluciones.

Por último, se mejoró la interacción entre los autos con imágenes puestas, con el fin que se viera más realista el comportamiento de la simulación.

Lunes 8 de junio:

Se necesitaba desesperadamente solucionar los errores del programa relacionado con las ventanas, se le preguntó a otro amigo y él te explicó que al meter las Ventanas en Threads, este puede generar muchos problemas.

Más tarde, se asistió a una tutoría con Santiago, con su ayuda por fin se entendió el otro error de las ventanas relacionadas con las imágenes y POO. Al parecer es que Tkinter no puede importar imágenes si se abren de más de 3 ventanas y estas ventanas están dentro de una clase.

Por lo que, se pensó lo siguiente como con 2 ventanas funcionaba y realmente lo que más importaba era la simulación del proyecto. Se tomó la idea de mi amigo de modificar constantemente la ventana principal del Menu, para que este realizará la animación.

Por último, ese día se le preguntó al profesor si necesariamente la animación debe ir antes de la ejecución de la simulación. Lo cual contestó afirmativamente. Entonces ese mismo día en la noche se programó la corrección de los errores.

Se le dedicó plenamente al desarrollo de la clase de la ventana Menu. Se investigó cómo guardar datos del programa en un archivo txt, además de obtener sus valores. Se crearon las diversas modificaciones de la ventana, al presionar un botón. Se creó la animación de las bolas rebotando en la pantalla. Además de dar aspectos estéticos al programa, para que se viera mucha más agradable a la vista.

Luego se modificó un poco del código de la Simulación para que utilizara los datos, guardados en el Taller del Automóvil.

Martes 9 de junio:

Más aspectos estéticos, como sonidos y una mejora de los métodos de cw-eb y choque, además de la creación de otro archivo py para usuarios de Mac o Linux. Como se encontró con un pequeño de error de colisión con la parte trasera del automóvil. Ese día se solventó el problema, dando así terminando la programación de la simulación.

Ese día se comenzó la documentación del proyecto. Introducción, descripción del problema y metodología.

Miércoles 10 de junio:

Se realizó la documentación de toda lógica del programa, ejemplos de corridas del programa y problemas que se tuvo durante el desarrollo del proyecto.

Jueves 11 de junio:

Cabe destacar que, durante la realización del proyecto se hicieron audios que hacían un resumen de lo acontecido ese día relacionado con el proyecto, estos audios posteriormente se convertirían en la bitácora del proyecto. Este día se dedicó a pasar de los audios de la bitácora a la documentación del proyecto.

Además de poner el tiempo realización de este y las conclusiones.

Tiempo de Realización del Proyecto

FUNCION	**TOTAL**
Requerimientos/diseño	14 horas
Investigación de funciones	16 horas
Programación	38 horas
Documentación interna	2 horas (Se realizaba mientras se programaba)
Pruebas	6 horas
Elaboración documento	8 horas
Corrección de errores	32 horas
TOTAL	115 horas

Conclusiones

Finalmente, de este proyecto se puden hacer varias conclusiones que son:

I. Como la tecnología nos puede ayudar a dar soluciones de la vida real.

II. Un mejor entendimiento del paradigma de programación de POO, ya que al principio de la realización del proyecto había cosas, que no se comprendían del todo, de este paradigma de programación.

III. El descubrimiento de muchas librerías que ayudaron al desarrollo de este, además del aprendizaje de muchos nuevos métodos y atributos de la librería Tkinter.

IV. El aprendizaje de la realización del primero proyecto de programación, como la importancia de documentar el código, el buen uso del tiempo, además de instruirse en la búsqueda por Internet para dar soluciones a errores o investigar sobre temas desconocidos.

Se espera que, con el conocimiento adquirido de este primer proyecto, se haga mucha más sencillo la planificación y elaboración de futuros proyectos.

Agradecimientos

Se le quiere agradecer a todas las personas que estuvieron involucrados en este proyecto, al profesor Jeff Smith por la ayuda de entendimiento del proyecto, a Nicolás Vigot por la ayuda respecto a la corrección de errores, a los tutores Santiago y Daniel. A mis amigos Issac, Javier, David y Ignacio, como las personas que me ayudaron como guía a la solución de errores y dudas acerca con Tkinter. Y a mi mamá que me apoyó emocionalmente durante el desarrollo del proyecto.