



DriiveR

Juan de la Cruz Caravaca Guerrero

Mayo 2022

Resumen

Documentación del desarrollo y características del programa informático "DriiveR" para la asignatura Desarrollo de Sistemas Hipermedia.

Índice

1	Introducción	3
2	Características	3
2.1	Menú principal	3
2.2	Pantalla de juego	3
3	Planificación	4
4	Herramientas utilizadas	5
5	Scripts	6
6	Problemas encontrados	6
6.1	Código de sincronización	6
6.2	Movimiento del vehículo	7
6.3	Diseño de niveles	7
6.4	COVID-19	7
7	Paquetes utilizados	8
8	Mejoras futuras	8
8.1	Uso de las señales	8
8.2	Diseño de niveles	8
8.3	Puntuaciones	8
8.4	Usabilidad	9
8.5	Portabilidad a otras plataformas	9
8.6	Realidad Aumentada	9

1. Introducción

DriiveR es un juego controlado por voz realizado íntegramente por Juan de la Cruz Caravaca Guerrero, alumno de Desarrollo de Sistemas Hipermedia 2021-2022 en la Universidad de Cádiz. El planteamiento es sencillo. En pantalla se muestra un coche circulando por una carretera. Imágenes de señales de tráfico van apareciendo, y el objetivo del usuario es decir su nombre. Las imágenes aparecen en puntos aleatorios de la pantalla, y a medida que el jugador progresá, se van haciendo más pequeñas y el tiempo de reacción también disminuye. La puntuación va aumentando conforme al tiempo que aguanta el jugador. Si el jugador falla o se queda sin tiempo, perderá una vida. Si no le quedan vidas, el juego termina, simbolizando un accidente.



Figura 1: DriiveR por la noche

2. Características

A continuación se describe el funcionamiento general del juego. El control de éste es exclusivamente por voz, no siendo necesario ningún otro controlador.

2.1. Menú principal

En el menú principal se muestran las posibles opciones. El usuario debe decir el nombre de cada una para seleccionarlas.

- **Iniciar** comienza una nueva partida de DriiveR.
- **Puntuaciones** muestra las puntuaciones obtenidas en partidas anteriores.
- **Ayuda** muestra una pantalla con la explicación general de los objetivos del juego.
- **Créditos** muestra un vídeo con los créditos del juego.
- **Salir** cierra la aplicación.

2.2. Pantalla de juego

A continuación se describe la interfaz del juego.

- Arriba a la izquierda se ven las **vidas** restantes.
- Arriba a la derecha se muestra el **tiempo** restante para responder correctamente la señal actual.



Figura 2: Menú principal



Figura 3: Pantalla de juego

- Abajo a la izquierda, el texto **"Salir"** indica que se puede volver al menú principal pronunciando dicha palabra.
- Abajo a la derecha se muestra la **puntuación** acumulada por el jugador.
- En medio de la pantalla aparecen las **señales** que el jugador debe decir.
A medida que el jugador progresá se produce un ciclo día-noche en el escenario. Además, la velocidad a la que se mueve el coche también aumenta.

3. Planificación

Para la realización de la aplicación inicialmente se decidió dividir el desarrollo en cuatro semanas. Se quiso utilizar la metodología de desarrollo ágil debido al escaso tiempo disponible y la familiaridad con este tipo de desarrollo.

- En la primera semana se plantearían las bases del diseño, se crearía el escenario básico y se empezaría a pensar en la lógica de programación.
- En la segunda semana se codificaría la mayoría de la lógica de la aplicación y se solventarían los problemas de implementación de las características planeadas.
- En la tercera semana se terminaría de programar la lógica del juego, se realizarían los menús y se trataría de introducir mejoras y detalles que fuesen factores de calidad.
- En la cuarta semana se terminarían los detalles, abandonándose las ideas que no hubiese sido posible añadir hasta el momento, y se realizaría la documentación.

Sin embargo la planificación no se ha cumplido. Aunque la primera semana empezó bien ya que suelo ser previsor, en la segunda enfermé (probablemente de COVID-19) y en la segunda y tercera semanas no hice casi nada, así que el grueso de la aplicación lo he hecho en la última semana.

4. Herramientas utilizadas

Desde un principio se deseó que el juego fuera bastante personalizado, para poder obtener más práctica con el uso de las herramientas disponibles. Las únicas herramientas utilizadas han sido:

- **Blender** para el modelado del vehículo y las animaciones. Se ha aprovechado lo que ya se tenía hecho de la entrega anterior, añadiendo pequeñas mejoras, como la matrícula.
- **Unity** para la realización general del juego
- **Unity ProBuilder**, para la realización de los objetos del mapeado. Al ser un juego en el que el vehículo, y por ende la cámara, se mueven a gran velocidad, no es necesario hacer un modelado demasiado complejo de los escenarios, por lo que esta herramienta es ideal. Sin embargo, se han podido comprobar las limitaciones de ésta, algunas de ellas de poca comprensión. Por ejemplo, las aristas de los modelos son muy poco visibles y los "tooltips."^{a1} al seleccionar las opciones de modelado aparecen con mucho retraso. Faltan opciones como eliminar aristas y vértidas y además, durante su uso, ocurrieron efectos inesperados en el modelado que no suceden al modelar con Blender.

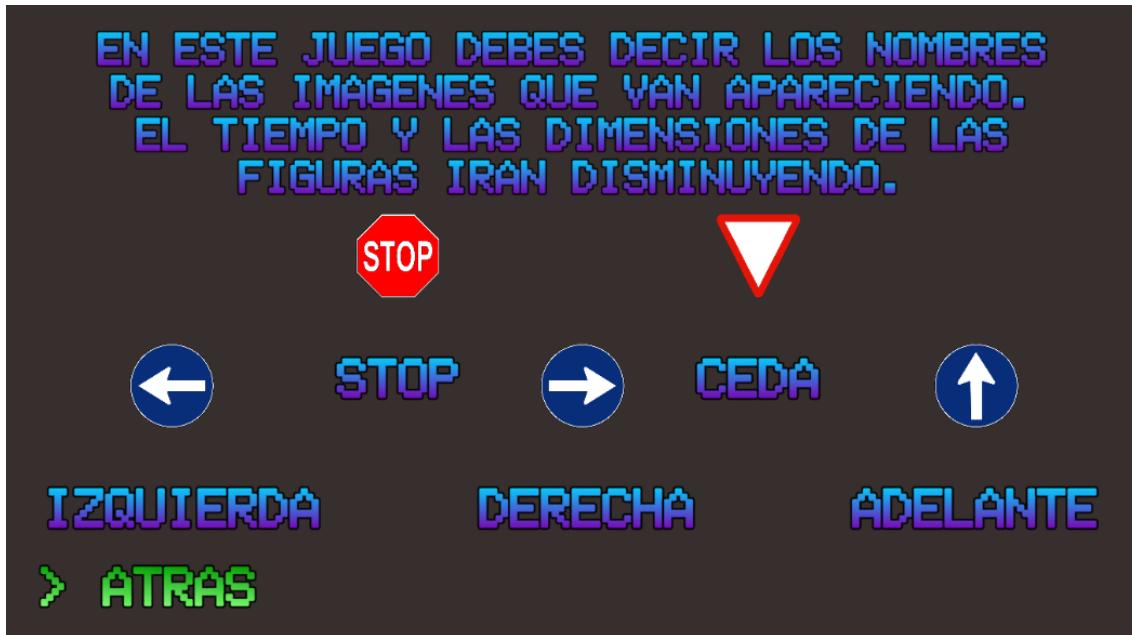


Figura 4: Pantalla de ayuda

5. Scripts

A continuación se describen de forma resumida los scripts que utiliza la aplicación. El código fuente de cada uno de estos scripts está debidamente comentado y documentado.

- **GameOverScript.cs** contiene el control para la escena de GameOver, específicamente mostrar la puntuación obtenida
- **LevelScrip.cs** contiene el código general del manejo de niveles y la aparición de imágenes
- **MainMenuObject.cs** es un útil script para desplazar los textos de las opciones de fuera a dentro del canvas, creando un llamativo efecto
- **MovableObject.cs** contiene el código para desplazar los objetos adelante y atrás (más información en "Problemas encontrados")
- **PostProcessScript.cs** contiene el código para manipular los efectos de postprocesamiento y crear el ciclo día-noche
- **Puntuaciones** contiene el código para recuperar y mostrar las puntuaciones obtenidas en su menú concreto
- **VoiceMainMenu.cs** contiene el código de reconocimiento de voz para las opciones del menú principal
- **VoicePuntuaciones.cs** contiene el código para el reconocimiento de voz en el menú de puntuaciones, aunque posteriormente se ha usado en otras escenas
- **SoundControl.cs** contiene el código para controlar la música en las escenas.



Figura 5: DriiveR por la tarde

6. Problemas encontrados

6.1. Código de sincronización

Una de las principales características que debía implementar la aplicación era el manejo del tiempo. Es necesario sincronizar los distintos elementos unos con otros. Las señales de tráfico permanecen un tiempo determinado en pantalla. Este tiempo es variable, y puede terminar de forma inesperada debido a un error del jugador. Además, en un primer momento se pensó en dejar un tiempo entre niveles, e incluso la aparición de señales de acierto o de error en base a las entradas

del jugador. Sin embargo, se terminó descartando esta idea debido a la complejidad de la misma y a lo poco que aportaba al resultado final.

Para la implementación del tiempo, previamente a la implementación se planeó utilizar corrutinas, debido al comportamiento de éstas, que se ejecutan a lo largo de múltiples frames. Estas corrutinas pausarían su ejecución el tiempo necesario. Sin embargo, también se descartó debido a la poca práctica con las mismas y la necesidad de sincronizarlas, optando por una implementación basada en las primitivas de tiempo de Unity y el control manual del tiempo.

6.2. Movimiento del vehículo

Debido a la naturaleza del juego en el que se muestra un coche avanzando hacia el infinito, era necesario idear una forma de representar esto. La solución fue sencilla: hacer que el coche avance a una determinada velocidad, y devolverlo al lugar de origen cuando se llegue a determinado punto. El mapeado y este punto "de retorno" deben estar bien definidos para que la transición sea suave y la impresión sea que el coche sigue avanzando normalmente. El resultado fue bastante satisfactorio. Este método es también utilizado por algunos juegos conocidos como Super Mario 64.



Figura 6: Las escaleras infinitas de Super Mario 64

6.3. Diseño de niveles

En un principio se planteó, para hacer más interesante el desarrollo, el realizar distintos mapeados por los que transcurriría el coche. El primero de ellos es el que utiliza la versión final, con la torre Eiffel de fondo. Desde aquí, el coche sería teletransportado mediante el método descrito en el anterior punto hacia otros mapeados. Otros escenarios planificados fueron las pirámides de Egipto, el Cristo Redentor de Río, el espacio exterior y la luna.

Sin embargo, esto entraría en conflicto con los scripts de postprocesamiento. Las diferentes ubicaciones requerían tonalidades de color diferentes, e idealmente skyboxes y pistas sonoras únicas. Esto habría de ser compatible con el ya existente manejo del ciclo día-noche, que se basa en cambiar la temperatura del color, lo cual suponía un problema. Además, también era necesario crear nuevos edificios y elementos que se viesen al avanzar. Tras determinar que haría falta demasiado tiempo y esfuerzo, se decidió descartar dicha característica, pero queda en el tintero para posteriores mejoras.

6.4. COVID-19

Otro problema no menos importante y que cabe la pena señalar es que en la segunda semana caí enfermo, probablemente de COVID-19, lo que me retrasó mucho en la realización del trabajo, por lo que al final no he podido dedicarle todo el tiempo que me hubiera gustado (aunque no deja de ser bastante).

7. Paquetes utilizados

El juego ha sido desarrollado en su mayoría sin el uso de paquetes externos. Se decidió así para poder obtener una mayor práctica en el uso de las herramientas de trabajo, especialmente Unity ProBuilder. Las únicas excepciones son:

- Paquete **PostProcessing** para el control de los efectos de postprocesamiento.
- Paquete **Standart Assets** para la utilización de objetos para decorar el terreno
- Paquete **Synthwave Music Pack** para la utilización de músicas de acompañamiento



Figura 7: Game Over

8. Mejoras futuras

8.1. Uso de las señales

Una característica interesante y que daría mayor solidez al juego sería que las señales que el usuario pronuncia tuvieran un mayor efecto en el juego. Así, si la señal fuera de "izquierda", el vehículo realizaría un giro hacia dicha dirección, si la señal fuera de "stop", el coche se pararía, y así sucesivamente. Esto daría un mayor realismo al juego, pero presenta varios problemas. El primero es que si el coche realizase un giro, el escenario debería cambiar, incluyendo los edificios y objetos de fondo. Pero la mayor complicación sería encontrar el punto del mapa en el que el coche debe girar. Además, ¿qué pasaría si el usuario dijese dos señales con poco tiempo de separación entre ellas? Esto requeriría un código de sincronización bastante complejo. Queda reservado para mejoras en un futuro.

8.2. Diseño de niveles

Como se mencionó en el apartado "Problemas encontrados", la realización de distintos niveles, pese a los problemas iniciales que presenta, se considera algo factible y que con seguridad será implementado en futuras actualizaciones.

8.3. Puntuaciones

Actualmente, las puntuaciones no son ordenadas de mayor a menor cuando son obtenidas o recuperadas del archivo en el que se encuentran. Este hecho, en principio, tiene una solución sencilla.

8.4. Usabilidad

Actualmente, sólo es posible el control por voz. Se planea la implementación de controles clásicos, aunque tan sólo para los menús.

8.5. Portabilidad a otras plataformas

La aplicación es dependiente del soporte de reconocimiento de voz de Windows, siendo necesaria la migración del código a otras plataformas si se quiere llevar la aplicación a ellas.

8.6. Realidad Aumentada

Se implementó el prototipo de una escena en la que el usuario debía escanear las distintas imágenes de señales de tráfico. Se descartó porque se decidió usar señales, las de dirección, cuya imagen es la misma pero con distinta rotación. Si las imágenes viniesen acompañadas de algún texto para diferenciarlas podría ser factible, y es una idea muy interesante a implementar.