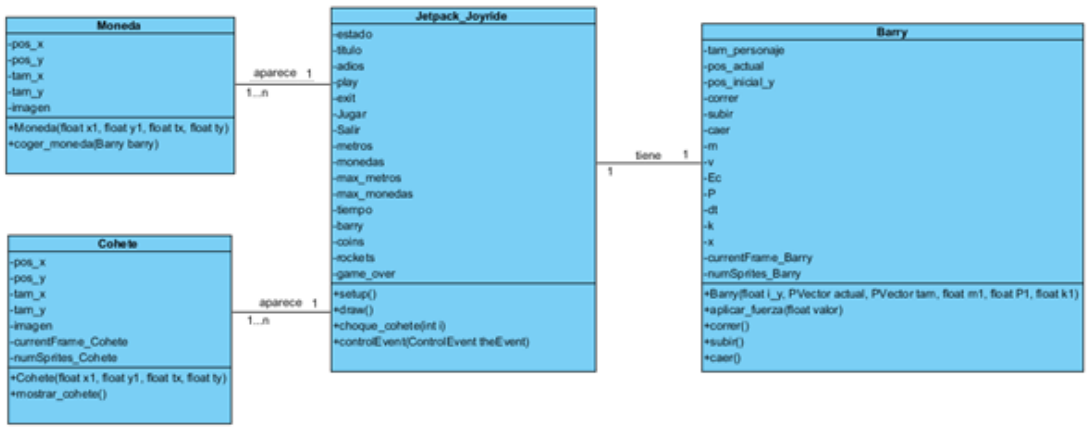
Jetpack Joyride

JOSE IGNACIO GARCÍA JÁVEGA

1. Introducción al problema

Este trabajo es un simple juego de scroll horizontal donde se controla la elevación de un personaje al tocar la pantalla, como en el juego original. Solo puede coger monedas y esquivar los cohetes letales mientras el hombre corre de manera infinita y se muestra los resultados obtenidos.

2. Diseño de clases realizado



Barry: es el individuo que controla el jugador. El método aplicar_fuerza usa las fórmulas estudiadas en simulación para aplicar aceleración en Barry, con algunas alteraciones:

Simulador de coche (rudimentario)

Ejemplo

Simularemos el acelerador de un coche.

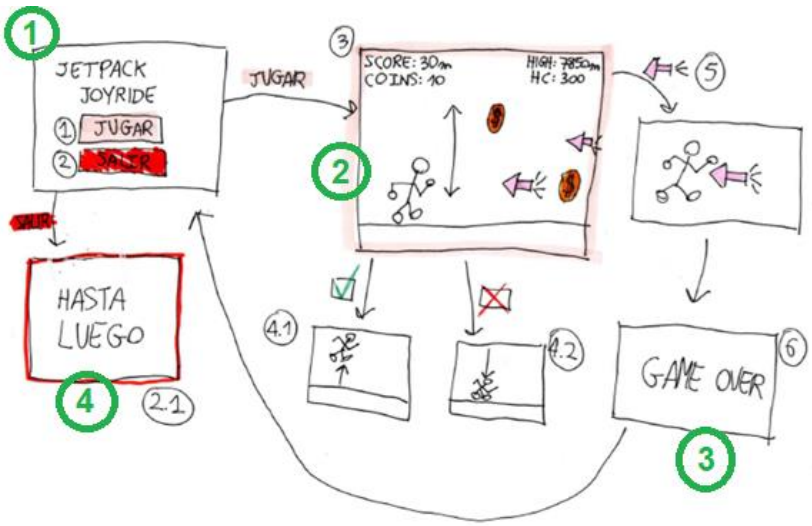
- Clase coche ($masa$, vel, E_c)
- $AplicarPotencia()$, actualizar $E_c = P\Delta t$ al pulsar una tecla.
- Pérdida de energía (rozamiento $F = -kv$) entonces $P = Fv = -kv^2$
- $updateVelo()$, actualiza la velocidad $v = \sqrt{2E_c/m}$

De este modo, se consigue desplazar a Barry en el eje Y.

Moneda y Cohete: son las clases que generan aleatoriamente las monedas que el jugador puede coger y los cohetes que se deben esquivar. Ambas clases tienen dos métodos que calculan la colisión con Barry (coger_moneda y choque_cohete), y que están basados en la función collide() del programa que está en el enlace https://processing.org/examples/bouncybubbles.html. Es importante aclarar que choque_cohete no está en la clase Cohete, sino en el programa principal, porque en caso de que Barry choque con uno, se acaba el juego y es recomendable destruir todos los componentes gráficos (que se crean en Jetpack_Joyride) para reiniciar correctamente el juego.

Jetpack_Joyride: es el programa principal que llama a las demás clases anteriormente mencionadas. También representa la máquina de estados del programa. Para entender mejor su funcionamiento, se muestra la imagen del boceto

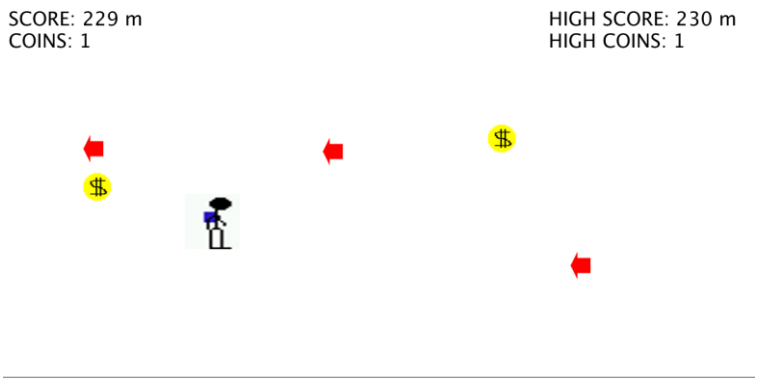
del plan de proyecto con los estados indicados con un número rodeado, todo de color verde:



1. Pantalla de inicio

Este menú principal compuesto con una imagen de fondo y dos botones que usan también imágenes aparece cuando se entra en la aplicación y cuando se acaba el juego.

2. Juego



Este es el juego que contiene los objetos generados de las clases Barry, Moneda y Cohete, y los textos donde se muestran los resultados.

3. Game Over

Esta simple imagen se muestra en un periodo de tiempo breve cuando el jugador colisiona con un cohete, mostrando los resultados que obtuvo al acabar la partida.

4. Salir

Esta pantalla de despedida es una simple imagen que se aplica de fondo durante poco tiempo antes de que se cierre el programa.

3. Pruebas y conclusiones

¿Cómo conseguí animar a Barry y los cohetes?

Hice un vector de Image para guardar los sprites que en conjunto realizan la acción de que Barry corra, caiga y se propulse; y que los cohetes se desplacen. Luego, para que se vieran todos los sprites, retoqué el valor de frameRate, junto con las variables para desplazar los cohetes y las monedas; y la potencia que aplicamos a Barry cuando sube y baja.

¿Cómo conseguí qué Barry colisionara bien con las monedas y los cohetes?

Primero hice que el fondo tuviera un color diferente al fondo de los sprites y coloqué una moneda o un cohete cerca de Barry para que se desplazara lentamente en los ejes X e Y desde cualquier dirección, para que se vieran bien como colisionaran. Luego usé los métodos de colisión y en cuanto colisionaran, mostraba un mensaje de contacto. Mientras tanto, retoqué los métodos hasta que las colisiones fueran satisfactorias, y en cuanto lo fueron, realicé los cambios deseados en las monedas y los cohetes (eliminarlos, incrementar el contador de monedas, terminar la partida…); además de dejar el fondo del juego del mismo color que el fondo de los sprites.

¿Cómo hice que Barry se propulsara?

Apliqué las fórmulas de aceleración estudiadas en simulación e hice aplicar_fuerza, pasándole un valor para aplicarlo a la potencia. Tanto a la hora de subir (que se consigue tocando la pantalla) como de bajar, le apliqué valores y desplazamientos diferentes, hasta obtener los resultados que me parecían idóneos teniendo en cuenta el valor del frameRate.

¿Cómo hice que aparecieran los estados 3 y 4 en poco tiempo?

Guardé en una variable el valor que se obtenía de second() y luego comparé el valor de second() con el de esa variable más un incremento pequeño. Ese incremento es el tiempo en que se muestran esos estados, y los ajusté de forma que se pudieran ver las pantallas el tiempo suficiente.

¿Cómo incrementé el contador de la distancia recorrida?

En un principio opté por una fórmula que partía de millis(), pero tras muchos intentos fallidos, decidí una técnica más simple: una variable entera que incrementa con el tiempo.

¿Cómo hice que funcionaran bien los botones del estado 1?

Al principio, en la función controlEvent realizaba los cambios a partir de los nombres que devolvía theEvent.getController(). Aunque funcionaba todo bien en el modo Java, al aplicar el programa en modo Android, tras pulsar los botones, el juego se cerraba enseguida sin saber por qué. Por tanto, en lugar de aplicar las acciones partiendo de los nombres, lo cambié por los labels; consiguiendo que el juego funcionará correctamente en los dos modos.

También aproveché y coloqué imágenes en los botones para que se vieran bien las palabras de éstos. Bastaba con cargar las imágenes como cualquiera, e introducirlas en los botones con setImage y setSize.

¿Por qué no cree una clase por cada estado?

Lo intenté con todos los estados cuando tenía claro que el programa funcionaba correctamente, pero al hacerlo me salían errores de los cuáles no sabía cómo resolver. Por tanto, decidí que los 4 estados se crearán en el mismo programa principal.

Conclusiones: este programa se podía haber enviado mucho antes, ya que se consiguieron cumplir con todos los requisitos antes de lo que se pedía. Sin embargo, se hicieron más pruebas para separar el programa principal en las 3 clases que están actualmente, mostrar bien todos los sprites de los objetos que hacerles parecer que se desplazan, conseguir que los botones cambiaran de estado… Aún así, el programa realiza correctamente todo lo que se describió en el documento work_plan. Algunos cambios que se podrían hacer para mejorar el juego son:

- Conseguir que el personaje no sobrepase la pantalla del dispositivo.
- Crear una base de datos para guardar las puntuaciones máximas.
- Colocar un botón en el estado 1 que permita al jugador borrar sus récords.
- Generar los cohetes y las monedas de manera más aleatoria.
- Tal vez, mejorar las colisiones.

En conclusión, se ha conseguido realizar a tiempo el programa como se acordó y con unos resultados muy convincentes. Para entender con mayor detalle el funcionamiento del programa, se puede mirar el código comentado junto con esta memoria.

Aclaración importante: es posible que no se muestre a la primera la aplicación al pulsar su icono. Basta con seguir pulsando el icono hasta que se consiga.