UNIVERSIDAD
DE GRANADA

Práctica 2
DragonBall Kart

Ingeniería de Software

MARCO ANTONIO RODRÍGUEZ MOLINA
DANIEL FERNÁNDEZ JIMÉNEZ

1 DE Junio DE 2024

Sistemas Gráficos

Ingeniería Informática

Descripción del Juego

Descripción inicial del juego:

- Nuestro juego se basa en Dragon Ball, por ejemplo el tubo que corresponde al circuito será con una textura que hará parecer una serpiente de la serie.
- El personaje principal será una nube con un niño/niña encima de la misma.
- Los obstáculos buenos serán bolas de dragón que proporcionan un aumento de velocidad temporal.
- Los obstáculos malos son una especie de estrellas oscuras que harán que la visión sea de un único color (por ejemplo verde), cambiando el color de la luz del escenario haciendo más difícil distinguir el escenario, durará 5 segundos en los que también se disminuirá la velocidad.
- Los objetos voladores serán planetas de distinto tipo que se diferenciarán ya sea por forma o color, los cuales al ser disparados dependiendo del tipo de este darán ventajas diferentes al personaje como pueden ser el quitar todos los efectos malos que se tengan en ese momento, un tiempo de inmunidad, etc…
- El modelo articulado será el propio personaje, al cual moverá los pies como si estuviera corriendo continuamente sobre la nube.
- El juego se jugará a contrarreloj de forma que habrá un límite de tiempo para llegar a la meta, en caso de que no se llegara a tiempo, la puntuación de este apartado será de 0 y en caso de hacerlo la puntuación por ello serán los segundos restantes que quedan de carrera al momento de cruzar la línea de meta.

Esto es un resumen de la propuesta inicial. Para verla entera con las imágenes que proporcionamos, hemos puesto en la carpeta de documentación el archivo inicial.

Correcciones realizadas por el profesor:

Dio el visto bueno a nuestra propuesta, sin embargo nos comentó que recordasemos:

- Aunque incluís variedad de objetos que seguro que cubren numerosas técnicas de modelado, debéis cumplir con el resto de los requisitos mínimos indicados en el guión de la práctica; en cuanto a animación, interacción, materiales, luces y cámaras.
- Uno de esos requisitos es que a cada vuelta completa del personaje en el recorrido, su velocidad se incrementará un 10%, con independencia de los cambios de velocidad que ya tenéis pensado según los obstáculos.

Modificaciones respecto a la propuesta inicial del juego:

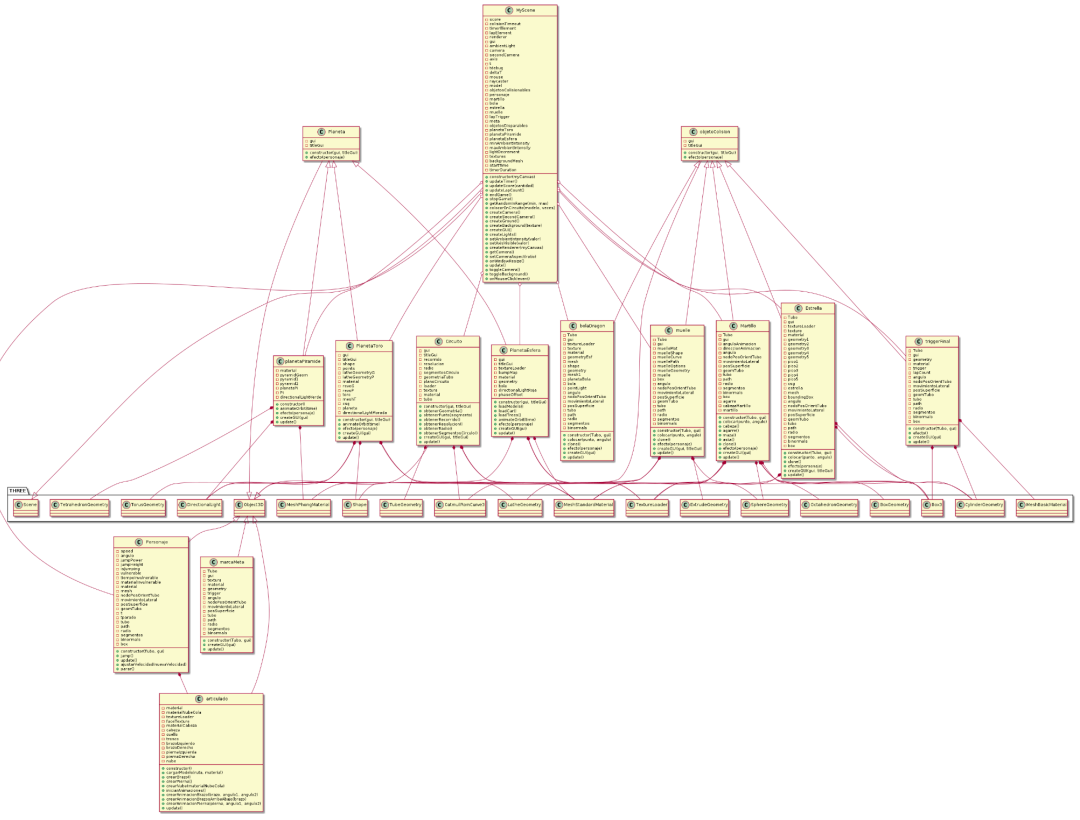
- El personaje articulado hace un movimiento constante de brazos (traslación y rotación) y de piernas (rotación). Además hemos incluido otro movimiento en la nube,

que se trata de una rotación de cada uno de los segmentos que componen la cola, lo que hace que se produzca un movimiento de vaivén (que va variando conforme pasa el tiempo).

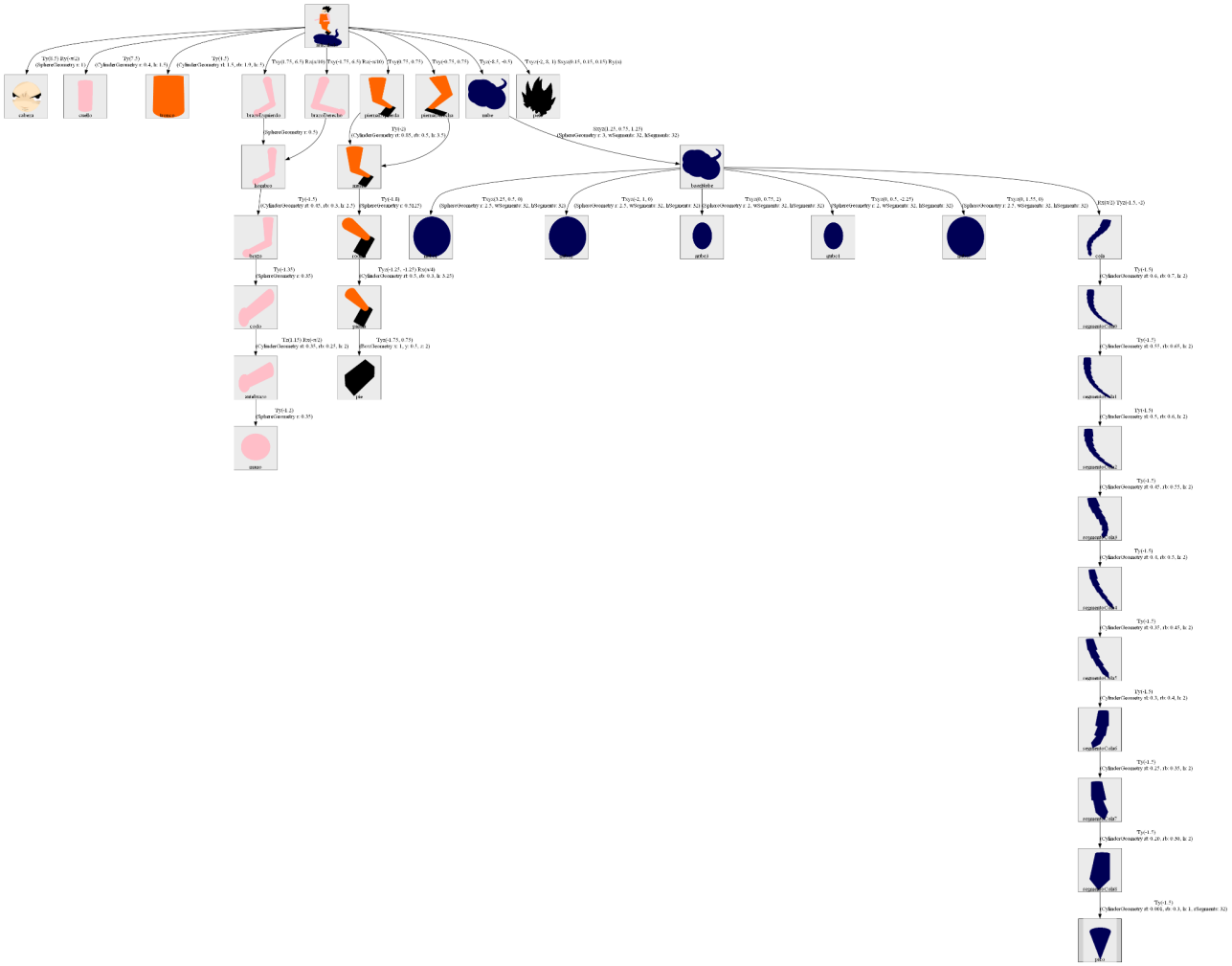
- Además hemos incluido un nuevo objeto, que es un muelle, que no suma puntos, simplemente cuando colisona el personaje, éste salta.
- Respecto a la iluminación, hemos considerado una disminución gradual de la luz ambiental, es decir mientras el personaje no coja una bola de dragón la luz ambiental irá disminuyendo. Sin embargo, si coges una estrella, la luz ambiental se pondrá a 0.
- También hemos introducido luces puntuales en las bolas de dragón, para que así destaquen más que los objetos malos. Y además cada planeta tiene una luz direccional del color del propio planeta.
- Hemos añadido un fondo, que es gradual, es decir, conforme la iluminación baja también cambia el fondo y al igual ocurre si la iluminación sube.
- Se ha tenido en cuenta el aumento de velocidad al completar cada vuelta.
- La puntuación también se ha modificado puesto que al tener en cuenta la dificultad del juego a la hora de encadenar rachas de colisiones buenas, hemos creído conveniente hacerla más simple. Puesto que de la otra manera se penalizaba mucho el fallo.

Diseño de la Aplicación

Diagrama de Clases.



Modelos Jerárquicos.



Cabe destacar que tanto el diagrama de clases como el modelo jerárquico los hemos incluido en archivos aparte para que se puedan ver con mejor calidad.

Algoritmos Importantes.

Métodos personajeArticulado.js (y de la clase colaReptil, contenida en js):

Dado que es el modelo articulado, creemos conveniente destacar sus métodos, aunque no se entrará en demasiado detalle ya que algunos pueden resultar muy simples.

- **crearBrazo():**
  Crea y devuelve un brazo articulado compuesto de un hombro, brazo, codo, antebrazo y mano. Cada parte se crea con geometrías y materiales específicos y se estructura jerárquicamente. Para que así la animación del personaje sea como se debe, ya que la rotación del hombro hace que rote todo lo que cuelga del hombro.

- **crearPierna():**
  Crea y devuelve una pierna articulada compuesta de un muslo, rodilla, pierna y pie. Al igual que el brazo, cada parte se crea con geometrías y materiales específicos y se estructura jerárquicamente. Del mismo modo el movimiento se hace con sentido.

- **crearNube(materialNubeCola):**
  Crea y devuelve un elemento compuesto de varias esferas de diferentes tamaños, utilizando un material específico, que es lo que hará de nube. Tiene una estructura jerárquica y también se le añade una cola reptiliana.

- **iniciarAnimaciones():**
  Inicializa las animaciones de los brazos y piernas del personaje. Crea movimientos de oscilación y desplazamiento para los brazos y piernas.

- **crearAnimacionBrazo(brazo, angulo1, angulo2):**
  Crea una animación de oscilación para un brazo, moviéndose entre dos ángulos especificados. Utiliza la biblioteca TWEEN para suavizar la animación y encadenar movimientos de ida y vuelta.

- **crearAnimacionBrazosArribaAbajo(brazo):**
  Crea una animación de desplazamiento vertical para un brazo, subiendo y bajando ligeramente. Utiliza TWEEN para suavizar la animación y encadenar los movimientos de subida y bajada.

- **crearAnimacionPierna(pierna, angulo1, angulo2):**
  Crea una animación de oscilación para una pierna, moviéndose entre dos ángulos especificados. Al igual que con los brazos, utiliza TWEEN para suavizar la animación y encadenar movimientos de ida y vuelta.

- **crearSegmentoCola(material, indice):**
  Crea y devuelve un segmento de la cola con una geometría cilíndrica y un material específico. La geometría se ajusta según el índice del segmento para reducir su tamaño progresivamente.

- **crearPico(material):**
  Crea y devuelve el pico final de la cola con una geometría cónica y un material específico. Para que así el último segmento de la cola sea en forma puntiaguda y punzante.

- **iniciarAnimacionCola(segmentos):**
  Inicializa las animaciones de todos los segmentos de la cola, creando movimientos de oscilación para cada uno.

- **crearAnimacionMovimiento(segmento, indice):**
  Crea una animación de oscilación para un segmento de la cola, moviéndose entre dos ángulos especificados. Utiliza TWEEN para suavizar la animación y encadenar movimientos de ida y vuelta. El movimiento de cada segmento se retrasa ligeramente en función de su índice para crear un efecto de ola. Con lo que se genera un movimiento de vaivén variado en el tiempo.

Métodos objetos:

- **Colocar(punto, angulo):**
  Función común a todos los objetos de la aplicación que necesitan posicionarse a lo largo del tubo. Esta función se encarga de posicionar el objeto a lo largo de la curva del tubo según un punto dado entre 0 y 1 y un ángulo de orientación entre 0 y 2PI. Sus principales responsabilidades son:
  - Posicionamiento a lo largo de la curva: Utiliza el método getPointAt del objeto path (que representa la curva) para obtener la posición en el espacio tridimensional correspondiente al punto dado a lo largo de la curva del tubo.
  - Orientación del objeto: Calcula la tangente a la curva en el punto dado utilizando el método getTangentAt. Esto ayuda a determinar la dirección hacia la cual debe estar orientado el objeto.
  - Orientación del objeto respecto al tubo: Ajusta la orientación del objeto para que esté alineado con la curvatura del tubo. Esto se logra configurando la propiedad up del objeto a la normal binormal de la curva en el punto específico.
  - Orientación adicional: Permite especificar un ángulo adicional para orientar el objeto alrededor del tubo además de la orientación natural proporcionada por la curva.

- **Clone():**
  Esta función también es común a todos los objetos y se utiliza para crear una copia exacta del objeto actual. Sus principales tareas son:
  - Creación de una instancia clonada: Instancia un nuevo objeto del mismo tipo que el objeto actual, pasando los mismos parámetros necesarios para su construcción, como la referencia al tubo y la interfaz gráfica.
  - Configuración de propiedades adicionales: Si el objeto tiene propiedades específicas que deben copiarse, como parámetros de configuración personalizados, estas también se configuran en la instancia clonada.
  - Retorno del objeto clonado: Retorna la nueva instancia del objeto, que puede utilizarse de forma independiente sin afectar al original.

- **animateOrbit(time):**
  Función que controla la animación de órbita alrededor de un centro en el espacio

tridimensional. Se encarga de animar el movimiento de los objetos planeta, cada uno orbitando alrededor de un eje. Aquí está su desglose:

- Configuración inicial:
  Se define el radio de la órbita, su velocidad y el centro de la órbita.
- Cálculo del movimiento:
  Basado en el tiempo pasado por parámetro, se calcula el ángulo de la órbita. Se utilizan funciones trigonométricas para encontrar las dos coordenadas variables del objeto en la órbita.
- Establecimiento de la posición:
  Las coordenadas calculadas junto con la posición del centro definen la nueva ubicación del objeto.

Métodos personaje:

- **document.addEventListener('keydown', (event):**
  Este bloque de código configura un listener para eventos de teclado, permitiendo controlar el ángulo del personaje y el salto a través de las teclas de flecha. Se ejecuta dentro del constructor de la clase Personaje y maneja los siguientes casos:
  - Flecha derecha (ArrowRight):
    Incrementa el ángulo del personaje (this.angulo) en 5 grados (equivalente a Math.PI / 36 radianes). Si el ángulo supera 2π (360 grados), se ajusta restándole 2π para mantenerlo en el rango [0, 2π).
  - Flecha izquierda (ArrowLeft):
    Decrementa el ángulo del personaje (this.angulo) en 5 grados.Si el ángulo es menor que 0, se ajusta sumándole 2π para mantenerlo en el rango [0, 2π).
  - Flecha arriba (ArrowUp):
    Si el personaje no está saltando (!this.isJumping), llama al método jump() para iniciar el salto.

- **jump():**
  Este método maneja la lógica del salto del personaje. Utiliza la librería TWEEN para crear una animación suave tanto en el ascenso como en el descenso.
  - Iniciar el salto:
    - Establece this.isJumping a true para indicar que el personaje está saltando.
    - Crea una animación (jumpTween) que incrementa la posición y del personaje (this.posSuperficie.position.y) hasta this.jumpHeight en 500 ms, utilizando una función de aceleración cuadrática para un efecto más natural.
  - Completar el salto:
    - Cuando el personaje alcanza la altura máxima, se inicia una animación de descenso (fallTween) que devuelve la posición y a su valor original (2.25) en 500 ms, también con una función de aceleración cuadrática.

- Finalizar el salto:
    - Cuando el descenso se completa, se restablece this.isJumping a false.

- **update()**
  Este método se llama en cada frame para actualizar el estado del personaje. Realiza varias tareas:
    - Actualizar TWEEN:
      Llama a TWEEN.update() para actualizar las animaciones en curso.
    - Actualizar la caja englobante:
      Recalcula la caja englobante (this.box) para reflejar la posición actual del personaje.
    - Gestionar el estado de velocidad y visibilidad:
        - Si la velocidad (this.speed) es 0, disminuye this.tparado hasta que llegue a 0, momento en el cual restablece la velocidad a 0.0005.
        - Alterna la visibilidad del personaje (this.visible) mientras this.tparado es mayor que 0 para dar un efecto de parpadeo en el personaje.
    - Actualizar la posición en el recorrido:
        - Incrementa this.t por this.speed y lo ajusta para que vuelva a 0 si supera 1. Esto indica el punto del tubo en el que deberá estar el personaje en el siguiente frame en base a la velocidad de este.
        - Calcula la nueva posición (posTmp) del personaje utilizando this.path.getPointAt(this.t) para obtener la posición en la curva correspondiente al valor actual de this.t y copia esta posición a this.nodoPosOrientTubo.position.
        - Obtiene la tangente de la curva en this.t usando this.path.getTangentAt(this.t). Ajusta posTmp añadiéndole la tangente para determinar la dirección hacia la cual debe mirar el personaje. Calcula el segmento actual en el que se encuentra el personaje y obtiene el binormal correspondiente.
        - Aplica la rotación lateral basada en this.angulo a this.movimientoLateral.
    - Gestionar el estado de invulnerabilidad:
        - Si el personaje es invulnerable (this.vulnerable es false), aumenta this.tiempoInvulnerable hasta que alcance 12, momento en el cual restablece la vulnerabilidad (this.vulnerable a true y this.tiempoInvulnerable a 0).

- **ajustarVelocidad(nuevaVelocidad):**
  Este método ajusta la velocidad del personaje:
    - Ajustar la velocidad:
        - Multiplica this.speed por nuevaVelocidad si el resultado está en el rango permitido (mayor que 0.0005 y menor que 0.005) y el personaje es vulnerable (this.vulnerable).

- **parar():**
  Este método detiene al personaje temporalmente:
  - Parar el personaje:
    - Si el personaje es vulnerable, establece this.speed a 0 y this.tparado a 10 para iniciar el estado de parada.

Métodos MiEscena:

- **updateTimer():**
  Actualiza el temporizador del juego, mostrando el tiempo restante en formato de minutos y segundos. En caso de acabarse el tiempo hace que la partida termine.

- **updateLapcount():**
  Actualiza el contador de vueltas del juego y finaliza el juego si el número de vueltas alcanza un límite.

- **endGame():**
  Este método finaliza el juego, calcula el puntaje final basado sumándole al actual el tiempo restante y lo guarda en el almacenamiento local del navegador, luego redirige al jugador a la página del leaderboard donde se mostrará dicha puntuación.

- **colocarEnCircuito(modelo, veces):**
  Este método coloca un modelo en el circuito del juego clonándolo varias veces en posiciones aleatorias y lo agrega a la escena, también añade los modelos clonados a una lista de objetos colisionables. Sirve para colocar alrededor del tubo objetos colisionables en posiciones aleatorias.

- **update():**
  El método update() se encarga de actualizar el estado del juego y la escena en cada frame. Sus principales funciones son:
  - Actualizar el tiempo:
    - Incrementa el contador de tiempo this.t usando this.deltaT que indicará el tiempo de animación de la órbita de los planetas.
    - Llama al método updateTimer() para actualizar el temporizador del juego de la interfaz.
  - Controlar el temporizador de colisión:
    - Disminuye this.colisionTimeout si es mayor que 0, lo que indica que aún no puede colisionar de nuevo con otro objeto hasta que este llegue a cero de forma que tras colisionar con un objeto este temporizador se activa para que no haya colisiones de más.
  - Actualizar el personaje:
    - Llama al método update() del objeto personaje.

- Actualizar la posición de objetos en órbita:
-
    - Recorre los hijos de la escena y llama al método animateOrbit(this.t) en los objetos que son instancias de Planeta.
- Detectar colisiones:
    - Recorre los objetos colisionables, llama a su método update() y verifica si colisionan con el personaje.
    - Si hay colisión y el tiempo de espera de colisión ha terminado, aplica los efectos definidos en el método efecto() de los objetos correspondientes. Además si el objeto es una instancia de triggerFinal que es el objeto que marca la meta, se suma una vuelta mediante updateLapCount(). En caso de que los objetos colisionados sean una bolaDragon o una estrella se actualiza la luz ambiental y el puntaje correspondiente a cada una.
- Decrementar la intensidad de la luz ambiental:
    - Disminuye gradualmente la intensidad de la luz ambiental si es mayor que la intensidad mínima.
- Cambiar la textura de fondo:
    - Cambia la textura de fondo según la intensidad de la luz ambiental que hay en ese momento aplicando texturas con más o menos brillo dependiendo de la situación.

- **onMouseClickEvent(event):**
  Se encarga de manejar los clics del ratón y detectar interacciones con objetos en la escena. Sus principales funciones son:
    - Actualizar las coordenadas del ratón:
        - Calcula las coordenadas this.mouse.x y this.mouse.y en el espacio de la pantalla, normalizándolas en el rango [-1, 1].
    - Configurar el raycaster:
        - Configura el this.raycaster para lanzar un rayo desde la cámara activa (this.activeCamera) hacia la posición del ratón en la escena.
    - Detectar intersecciones:
        - Utiliza el raycaster para detectar intersecciones entre el rayo y los objetos disparables (this.objetosDisparables) que serán los tres planetas de la escena.
    - Procesar la intersección:
        - Si hay intersecciones, encuentra el objeto seleccionado y verifica si es una instancia de Planeta.
        - Si es así, hace que el objeto seleccionado sea invisible y aplica el efecto correspondiente definido para cada tipo de planeta llamando a efecto(this.personaje).
        - Actualiza el puntaje sumando 100 puntos.
    - Hacer visible el objeto después de 20 segundos:

- Utiliza setTimeout para volver a hacer visible el objeto después de 20 segundos.

Material Descargado de Internet

Se han utilizado recursos descargados de internet en el desarrollo del juego, incluyendo modelos 3D, texturas. A continuación se detallan las referencias:

Modelos 3D

Se han usado los siguientes modelos 3D para la creación de los distintos objetos de la escena:

- hair.obj: https://sketchfab.com/3d-models/goku-dragon-ball-hair-free-model-anime-style-7bb0bda73676466aa4bb5b66a040102e
- Low-Poly-Racing-Car.obj: https://free3d.com/3d-model/low-poly-racing-car-22092.html
- Lowpoly_tree_sample.obj: https://free3d.com/3d-model/low_poly_tree-816203.html

Manual de Usuario

¿Cómo jugar?

- Tu personaje deberá completar 3 vueltas antes de que se acabe la cuenta atrás de dos minutos y conseguir la puntuación más alta posible. Para ello, puedes moverte con la flecha izquierda y derecha a lo largo de la serpiente (el circuito).
- Has de saber que puedes toparte con objetos buenos, objetos malos y objetos neutros, que se explican sus efectos más adelante.
- También podrás disparar a unos planetas voladores que causarán efectos buenos sobre ti.
- Ten cuidado con la luz porque se va apagando poco a poco y te quedarás a oscuras, deberás coger un elemento concreto para remediarlo.
- Puedes cambiar de cámara pulsando la barra espaciadora. Recomendamos usar la cámara lejana para disparar y la cámara en tercera persona para guiar al personaje.
- Si quieres jugar sin fondo ambientado puedes quitarlo pulsando la tecla del punto (pero pierde encanto, ya que el fondo varía con la iluminación).
- El juego termina, bien completando las 3 vueltas o que se acabe la cuenta atrás.
- Si se completan las 3 vueltas, el tiempo restante de la cuenta atrás se sumará a la puntuación ya acumulada.
- Al finalizar el juego te mostrará tu puntuación y te dará la opción de volver a jugar.

Descripción de cómo afectan los elementos del juego al personaje.

Elementos “neutros”.

Este elemento es un muelle, el cual hace que el personaje salte si colisiona.

- Puede considerarse bueno, puesto que avanza durante un momento más rápido (al saltar) y/o porque puedes esquivar elementos malos (saltando por encima).
- Hay que tener especial cuidado con saltar cerca de la meta, puesto que si se salta por encima del aro que delimita la vuelta, no contará como vuelta completada y perjudica al jugador puesto que ha de hacer una vuelta completa añadida (con el gasto de tiempo que esto conlleva).
- No afecta a la puntuación.

Elementos a atrapar y sus ventajas.

El elemento a atrapar son las bolas de dragón, ya que son totalmente buenos para el jugador.

- Aumentan la velocidad un 2%.
- Devuelven la iluminación ambiental al máximo.
- Suman 15 puntos.
- Están iluminadas con una luz puntual (fácil encontrarlas cuando la iluminación sea escasa)

Elementos con los que no colisionar y consecuencias.

Existen dos tipos de elementos de este tipo: las estrellas y los martillos.

Las estrellas:

- Disminuyen la velocidad un 2%.
- Ponen al mínimo la iluminación ambiental (es decir a oscuras) hasta que se recoja una bola de dragón.
- Restan 15 puntos.
- No están iluminadas con lo que en la oscuridad, apenas se ven, es fácil colisionar con ellas.

Los martillos:

- Al colisionar con la cabeza del martillo (“te aplasta”), el personaje se queda quieto unos segundos, lo que hace que el tiempo corra en tu contra.
- Te devuelven a la velocidad base. Los aumentos que hayas podido acumular, tanto de completar vueltas, como de recoger bolas de dragón o disparar planetaPiramide se neutralizan.
- No afecta a la puntuación.

Consecuencias de disparar a objetos voladores.

- Cada planeta, independientemente del tipo que sea, al dispararle suma 100 puntos. Por contraposición, aportan luz direccional (de diferentes colores), hacen que aunque la luz ambiente sea nula se vea aunque sea en colores. Así pues si le disparas esa luz direccional no la podrás tener para orientarte.

- A los 20 segundos reaparece el planeta que ha sido disparado.

- Existen tres tipos de planetas diferentes, que aparte de sumar puntuación tienen los siguientes efectos sobre el jugador:

    - planetaEsfera (luz direccional roja): hace al personaje invulnerable unos segundos, es decir no le afectan los objetos malos (estrellas y martillos) con los que colisiones.
    - planetaToro (luz direccional morada): en caso de que se encuentre paralizado bajo el efecto de un martillo, este lo desparaliza instantáneamente.
    - planetaPiramide (luz direccional verde): aumenta la velocidad del personaje al doble de lo que en ese momento posea. Conviene disparar cuando el personaje haya colisionado con un par de bolas de dragón seguidas o completado vuelta.