Grado en Ingeniería Informática

INFORMÁTICA GRÁFICA Y VISUALIZACIÓN

Proyecto Final. Pinball

Alumno: Sergio Perea de la Casa.

DNI: 77433569-K.

Correo: spc00033@red.ujaen.es

Profesor: Francisco Daniel Pérez Cano.

Grupo de prácticas: Grupo 3.

Horario: Jueves (12:30 - 14:30).

Para la obtención del proyecto completo, es necesario acceder al siguiente enlace con cuenta asociada a la Universidad de Jaén. En él aparece una carpeta comprimida con el código fuente y las imágenes de las texturas usadas (PINBALL_codigo_fuente.zip) y otra carpeta comprimida con todo el proyecto en caso de tener algún problema con únicamente el código fuente.

https://drive.google.com/drive/folders/1YEc-D5H1lw1h1qJ9GsmZ1IT6Ulz0mno_?usp=sharing

El siguiente enlace muestra un vídeo descriptivo sobre el proyecto (y otro con cortes para acortar el vídeo) donde hablo sobre algunas anotaciones en él para indicar cómo es el movimiento en el escenario, su interacción con él y la posible activación (pausa/reanudación) del juego.

Es necesario entrar mediante una cuenta google asociada a la Universidad de Jaén.

https://drive.google.com/drive/folders/1UZxgIXfZ43lSend3gmADaw5-y_nNq6iE?usp=sharing

INTRODUCCIÓN

Para dar una breve introducción sobre mi proyecto voy a centrar la introducción en una breve explicación del concepto del proyecto y mis objetivos a conseguir en él.

El proyecto consiste en realizar un PINBALL, videojuego conocido desde hace mucho años con una mecánica aparentemente sencilla a ojos del jugador. En él, el usuario tiene un tablero, donde una bola va cayendo hacia sus palancas mediante una velocidad que puede ir variando (dependiendo de los rozamientos y fuerza con la que se ha movido anteriormente la bola). Una vez el usuario ve que la bola está apunto de entrar en el hueco donde se encuentra sus palancas debe de realizar unos movimientos de las palancas para desplazar a la bola en dirección contraria. El jugador, en caso de no conseguir darle a la bola, ésta se reinicia a la posición más lejana de las palancas y se vuelve a empezar el proceso.

Los objetivos que he planteado y los que he conseguido plasmar son, como es normal, la creación de un escenario similar a un tablero de pinball. En él, se crea un obstáculo centralizado en la anchura del tablero y algo desplazado en profundidad hacia el extremo opuesto a las palancas.

Otro objetivo es el posible movimiento de las palancas, de forma que estén limitadas a una rotación tanto máxima como mínima y así simular el movimiento que tienen real las palancas de un pinball físicamente hablando.

Como **objetivo central**, y el que más dificultad lleva, es el movimiento de la bola por el tablero. Mi objetivo con ello no era mantener la bola a una misma velocidad y simplemente ir cambiando su dirección respecto al tipo de choque que se produjese con ella. Mi objetivo con este movimiento era darle realismo; es decir, partiendo de una velocidad inicial igual a 0 y sabiendo que la aceleración del peso de un cuerpo es |a| = 9.8 (pero sin tener en cuenta un peso de la bola) entonces he conseguido realizar diferenciación de movimientos y pérdida de velocidad gracias a este sistema.

Como objetivo de iluminación he pensado realizar una iluminación focal hacia el tablero con toque rojizo y una iluminación por la “ventana” del escenario amarillenta que ilumine la habitación y al tablero.

DISEÑO

Para el diseño de mi implementación se ha generado unas clases adicionales para su uso modulado en clases generales para la visualización de una escena como son las siguientes:

Clases orientadas a objetos

Se va a hacer un recorrido general de cada una de las clases más importantes y su utilización significativa en el proceso de diseño. Se marcará **en negrita** los atributos/métodos que considero más interesantes.

Bola

Una clase importante para la implementación de un juego como es el pinball. En ella, se crea una bola mediante figuras de OPEN_GL. Pero lo necesario de su modulación y creación de una clase para ello es los atributos/métodos listados a continuación:

- Posición de la bola: *float posX, posY, posZ.*
- Direcciones en ejes X y Z: *unsigned int dirX, dirZ.*
- Movimientos de velocidad/aceleración: *float velocidad, aceleración.*
- **Métodos de cambio de velocidad:** *cambioVelocidadPalancas(...), cambioVelocidad(...).* El segundo método hace referencia al cambio de velocidad por colisión con el escenario sin corresponder a las palancas, ya que estas añaden una velocidad extra a la velocidad.
- Getters, setters, animación y visualización.

Palanca

Clase que referencia a la creación de objetos vinculados al objeto que interactúa con el objeto bola. En él salen una serie de atributos y métodos a destacar:

- Longitud de la palanca: *float longitud.*
- Material que construye la palanca: *GLUquadricObj * cilindro.*
- **Grados de movimiento (y limitaciones) que tiene la palanca:** *float gradosMAXrot, float gradosMINrot, rotPalanca.*
- Indicadores de estar subiendo, bajando y otros: *bool subiendo, bajando, llegadoLimiteMAX, llegadoLimiteMIN.*
- **Métodos de activación de movimientos de subida y bajada:** *getSubiendo_Palanca(), setSubiendo_Palanca(...), getBajando_Palanca(), setBajando_Palanca(...)*.
- Métodos de comprobaciones de llegar a límites de rotaciones, movimientos de rotaciones y más.

Clank

Clase que ayuda a limpiar el código de la clase Escena3D, ya que el grafo de escena del muñeco mantiene una serie de métodos para su construcción bastante significativa. Los métodos son los mismos implementados en la práctica del grafo de escena.

igvCilindro

Clase hija de igvMallaTriangulos la cual sirve para añadir algunos modelos en la escena a partir de una malla de triángulos como son las pendientes de las columnas para ayudar a la bola a llegar a las palancas o la plataforma del muñeco “Clank” para su posible animación en una rueda giratoria.

A continuación, las siguientes dos clases son las clases generales del diseño de mi proyecto. En ellas, se desarrolla toda la animación e interacción con ellos.

igvInterfaz

La clase mantiene toda configuración sobre el manual de usuario, el cual se explica en el apartado de ANEXO. Mantiene toda animación e interacción con la escena. Como atributos/métodos a destacar:

- **Tiempo:** *float tiempo.* **Atributo muy importante** para el proceso de realismo de la bola sobre el tablero.
- Dirección en el eje X: *int dirX.* Este atributo ha sido creado para solventar problemas de direccionamiento a la hora de producirse colisiones.
- **Atributos de activación de nuevos eventos/ cambios de valores repentinos:** *bool tocaTecho, tocaPlataformaClank_baja, tocaPlataformaClank_alta, modificado, modificadoPalDer, modificadoPalIzq.*
- **Valor añadido por choque de bola:** *float bolaChocadaMovX.* Este atributo es el que aumenta el movimiento hacia una dirección o hacia otra, dependiendo del choque.
- **Métodos de animaciones:** *animacionBola(), animacionClank(), subirPalancaIzq(), subirPalancaDer().* Estos métodos tendrán una breve explicación en el apartado de implementación sobre qué animación y cómo se produce.
- Método de cambio de vista: *cambiarVista().*

igvEscena

Como última clase a destacar, en ella se crea el modelo con ayuda de las anteriores clases (Bola, Clank, Palanca) que acabará siendo visualizado. Además en él existen una serie de atributos/métodos a destacar:

- **Atributos de posicionamiento:**
  - *float focoX, focoY, focoZ* (correspondientes a la posición de los focos en el tablero, con iluminación rojiza).
  - *float posPlatX, posPlatY, posPlatZ* (correspondientes a la posición del objeto que obstaculiza y provoca diferentes direcciones en el escenario).
  - *float anchura, profundidad, altura* (correspondientes a las dimensiones del tablero del pinball).

- **Atributos para texturas**: *bool texturaCargada, vector<igvTextura*> texturas.* Controlando con el booleano que no se carguen cada vez que se llama al método visualizar.
- Todos l**os demás atributos a destacar** corresponden a la creación de **objetos** de las **clases explicadas anteriormente**.

- **Métodos para pintar el escenario:** *pintar_suelo_techo(...), pintar_paredes(...), pintar_tablero(...), pintarPatas_tablero(), accesoriosEscenario(), bordesEscenario(...), crealluminacion(...).*

- **Métodos de creación de muros con pendiente en el escenario:** *trianguloInfDer(...), trianguloInfIzq(...)*.

IMPLEMENTACIÓN

En este apartado voy a referirme a los métodos de implementación, sobre todo, en la clase igvInterfaz ya que creo que es el más importante implementado en mi proyecto.

igvInterfaz::animacionBola()

Método basado en el movimiento de la bola y en diferentes interacciones que provocan su cambio de dirección. La estructura es la siguiente:

1. **Evento en caso de estar la bola en posición para darle con las palancas:** Dependiendo de donde se encuentre la bola, para darle con las palancas (posición de la palanca izquierda, derecha o centro) se ejecuta una serie de cambios de velocidad en la bola y en caso de darse con la palanca izquierda o derecha entonces se cambia la dirección en el eje x.

```
/*JUSTAMENTE CENTRO*/
if (interfaz.escena.getBola().getPosX() >= -0.05 && interfaz.escena.getBola().getPosX() <= 0.05) {
    if (interfaz.escena.getPalIzq().getSubiendo_Palanca() && interfaz.escena.getPalDer().getSubiendo_Palanca()) { /*Si la palanca además se encuentra subiendo...*/
        interfaz.escena.getBola().cambioVelocidadPalancas(interfaz.escena.getPalIzq().getPalancaRot(), interfaz.escena.getPalIzq().getPalancaRotMAX(), interfaz.tiempo);
        modificado = true;
    }
}
/*PARTE DE LA DERECHA DEL TABLERO*/
if (interfaz.escena.getBola().getPosX() >= 0.0 && interfaz.escena.getBola().getPosX() < interfaz.escena.getPalDer().getLongitud() && !modificado) {
    if (interfaz.escena.getPalDer().getSubiendo_Palanca()) { /*Si la palanca además se encuentra subiendo...*/
        interfaz.escena.getBola().cambioVelocidadPalancas(interfaz.escena.getPalDer().getPalancaRot(), interfaz.escena.getPalDer().getPalancaRotMAX(), interfaz.tiempo);
        modificadoPalDer = true;
        interfaz.dirX = -1;
    }
}
/*PARTE DE LA IZQUIERDA DEL TABLERO*/
if (interfaz.escena.getBola().getPosX() <= 0.0 && interfaz.escena.getBola().getPosX() > -interfaz.escena.getPalIzq().getLongitud() && !modificado) {
    if (interfaz.escena.getPalIzq().getSubiendo_Palanca()) { /*Si la palanca además se encuentra subiendo...*/
        interfaz.escena.getBola().cambioVelocidad(interfaz.escena.getPalIzq().getPalancaRot(), interfaz.escena.getPalIzq().getPalancaRotMAX(), interfaz.tiempo); //Se ca
        modificadoPalIzq = true;
        interfaz.dirX = 1;
    }
}
```

2. **Eventos de chocar con el borde del escenario:** Estos eventos producen sobre todo un cambio de dirección basado en el cambio de valores la asignación correspondiente de valor en el atributo interfaz.dirX y añadiendo un valor de movimiento por bolaChocadaMovX, el cual se ha explicado anteriormente.

```
/*CHOCAR CON LA PARED DE LA IZQUIERDA*/
if (interfaz.escena.getBola().getPosX() <= -interfaz.escena.getAnchura() + 0.2) {
    interfaz.dirX = 1;
    interfaz.bolaChocadaMovX = 0.1;
}

/*CHOCAR CON LA PARED DE LA DERECHA*/
if (interfaz.escena.getBola().getPosX() >= interfaz.escena.getAnchura()- 0.2) {
    interfaz.dirX = -1;
    interfaz.bolaChocadaMovX = -0.1;
}

/*CHOCAR CON EL "TECHO"*/
if (interfaz.escena.getBola().getPosZ() <= -interfaz.escena.getLongitud() + 1 && !tocaTecho) {
    interfaz.escena.getBola().setVelocidad(-interfaz.escena.getBola().getVelocidad()/2);
    interfaz.tiempo = interfaz.tiempo / 2;
    tocaTecho = true;
}
else if (interfaz.escena.getBola().getPosZ() > -interfaz.escena.getLongitud() + 1.01 && interfaz.escena.getBola().getPosZ() < -interfaz.escena.getLongitud() + 1.2) {
    tocaTecho = false;
}
```

3. **Eventos de chocar con los muros en pendiente:** Para ello se ha tenido en cuenta, según en la posición en el eje x de la bola respecto al muro correspondiente cuánta de velocidad es necesaria según un en la parte de la pendiente en la que se encuentra. Mostraré el choque con el muro derecho, siendo para el izquierdo igual pero con la inversa al valor dado a interfaz.dirX.

```
/*CHOCAR CON EL MURO INFERIOR DERECHO:
* Para intervalo [anchura-1,anchura]
* Para intervalo [anchura-2,anchura-1]
* Para intervalor [anchura-3,longitud Palanca derecha]
*/
if (interfaz.escena.getBola().getPosX() > (interfaz.escena.getAnchura() - 1)  && interfaz.escena.getBola().getPosZ() >= (interfaz.escena.getLongitud() / 3 )) {
    interfaz.escena.getBola().cambioVelocidad(27, 45, interfaz.tiempo);
    if (interfaz.dirX != -1) interfaz.dirX = -1;
}
else if (interfaz.escena.getBola().getPosX() > (interfaz.escena.getAnchura() - 2) && interfaz.escena.getBola().getPosZ() >= (interfaz.escena.getLongitud() / 3 + 0.5)) {
    interfaz.escena.getBola().cambioVelocidad(25, 45, interfaz.tiempo);
    if (interfaz.dirX != -1) interfaz.dirX = -1;
}
else if (interfaz.escena.getBola().getPosX() > (interfaz.escena.getAnchura() - 3) && interfaz.escena.getBola().getPosZ() >= (interfaz.escena.getLongitud() / 3 + 1)) {
    interfaz.escena.getBola().cambioVelocidad(20, 45, interfaz.tiempo);
    if (interfaz.dirX != -1) interfaz.dirX = -1;
}
```

4. **Evento de choque con el objeto de la plataforma del muñeco animado Clank:** Este evento es el más complejo en código debido a su parametrización de que dependiendo de donde se encuentre, se sigan produciendo los choques aparentes. Se está controlando el choque en la semicircunferencia de la izquierda por un lado y la semicircunferencia de la derecha por otro. Muestro la implementación llevada por la parte de la izquierda y derecha de la parte de abajo de la plataforma siendo la parte superior bastante similar.

```
/*PARTE IZQUIERDA ABAJO*/
if (interfaz.escena.getBola().getPosX() < interfaz.escena.getPlat_posX()
    && interfaz.escena.getBola().getPosX() > interfaz.escena.getPlat_posX() - interfaz.escena.getPlataformaClank().getRadio()
    && interfaz.escena.getBola().getPosZ() > interfaz.escena.getPlat_posZ()
    && interfaz.escena.getBola().getPosZ() < (interfaz.escena.getPlat_posZ() + interfaz.escena.getPlataformaClank().getRadio() + 0.1) && !tocaPlataformaClank_baja) {
    interfaz.escena.getBola().setVelocidad(-interfaz.escena.getBola().getVelocidad()/2);
    interfaz.tiempo = interfaz.tiempo / 2;
    interfaz.dirX = -1;
    interfaz.bolaChocadaMovX = interfaz.dirX * 0.1;
    tocaPlataformaClank_baja = true;
}
/*PARTE DERECHA ABAJO*/
else if (interfaz.escena.getBola().getPosX() >= interfaz.escena.getPlat_posX()
    && interfaz.escena.getBola().getPosX() < interfaz.escena.getPlat_posX() + interfaz.escena.getPlataformaClank().getRadio()
    && interfaz.escena.getBola().getPosZ() > interfaz.escena.getPlat_posZ()
    && interfaz.escena.getBola().getPosZ() < (interfaz.escena.getPlat_posZ() + interfaz.escena.getPlataformaClank().getRadio() + 0.1) && !tocaPlataformaClank_baja) {
    interfaz.escena.getBola().setVelocidad(-interfaz.escena.getBola().getVelocidad() / 2);
    interfaz.tiempo = interfaz.tiempo / 2;
    interfaz.dirX = 1;
    interfaz.bolaChocadaMovX = interfaz.dirX * 0.1;
    tocaPlataformaClank_baja = true;
}
/*REESTABLECER EL CHOQUE EN LA PARTE DE ABAJO*/
else if (interfaz.escena.getBola().getPosZ() > (interfaz.escena.getPlat_posZ() + interfaz.escena.getPlataformaClank().getRadio() + 0.101))
    tocaPlataformaClank_baja = false;
```

5. **Actualización de posición de la bola, en los ejes X y Z, y de su velocidad / tiempo:** A destacar de ello es que para evitar desplazamientos extraños en el eje X, no se incluye en concepto de velocidad a la suma en su desplazamiento. En vez de ello, se sustituye por el atributo de choque interfaz.bolaChocadaMovX. También es importante destacar el concepto de que cuando pierde toda velocidad debido a la aceleración es necesario restablecer el atributo interfaz.tiempo para que vuelva a tener una similitud a caída libre tras su pérdida completa de velocidad en la bola. Por último, la actualización de la velocidad que va llevando la bola y el tiempo que se va transcurriendo para el posible incremento de velocidad en la bola.

   /*ACTUALIZACIÓN DE LA POSICIÓN Z*/
   ```
   interfaz.escena.getBola().setPosZ(interfaz.escena.getBola().g
   etPosZ() +
   interfaz.escena.getBola().getVelocidad()*interfaz.tiempo +
   (interfaz.escena.getBola().getAceleracion()*pow(interfaz.tiem
   po,2)/2));
   ```

   /*ACTUALIZACIÓN DE LA POSICIÓN X*/
   ```
   interfaz.escena.getBola().setPosX((interfaz.escena.getBola().
   getPosX() + interfaz.dirX * (
   interfaz.escena.getBola().getAceleracion() *
   pow(interfaz.tiempo, 2) / 2) + interfaz.bolaChocadaMovX ));
   ```

   /*ACTUALIZACIÓN DE LA VELOCIDAD*/

   ```
   interfaz.escena.getBola().setVelocidad(interfaz.escena.getBol
   a().getVelocidad() +
   interfaz.escena.getBola().getAceleracion() *
   interfaz.tiempo);
   ```

   /*INCREMENTO DEL "TIEMPO*/
   ```
   interfaz.tiempo += 0.0003;
   ```

6. **Evento de reinicio de la partida:** Una vez el jugador pierde por no haber dado bien a la bola, se restablecen los valores que tenía el inicio del juego.

PROBLEMAS ENCONTRADO Y SOLVENTADOS

En este apartado voy a expresar cuáles han sido los problemas a gran escala que he tenido durante el proyecto y como he solucionado dicho problema.

El primer problema me surge respecto al **uso de texturas** de forma que por cada vez que se llame al visualizar de la escena no se cree de nuevo las texturas. Esto es muy ineficiente y se ve reflejado en el movimiento de la bola. Para solucionar este problema he usado un condicional que indique si ya se habían cargado anteriormente mediante un booleano. También, para poder usar diferentes modos de texturas, una vez indico visualización de la textura paso como parámetro un número que me indica si quiero que sea GL_MODULATE, GL_REPLACE, etc.

El segundo problema me surge en el posible **movimiento de la bola de forma realista**, ya que tras querer usar una fórmula parecida al movimiento de una bola muchas de las variantes de su movimiento en las colisiones provocaba que la bola se descontrolara o provocara unos movimientos después de la colisión poco realistas. Como he explicado en la implementación, su solución a ello ha sido añadir una serie de atributos que me sirvieran de apoyo para cambiar bien su dirección.

El mayor problema en el proyecto ha sido el tiempo que he tenido para gestionar todas las prácticas a entregar respecto a todas las prácticas. Debido a que se ha aplazado toda última entrega a Enero y sabiendo que los exámenes son en estas fechas no he podido añadir muchas de las funcionalidades que tenía pensado añadir como por ejemplo una selección de la plataforma que sirve de obstáculo para poder moverla por el escenario cuando no se esté jugando.

CONCLUSIONES

Una vez empecé con la asignatura, realmente no tenía ningún concepto sobre la implementación gráfica de objetos ni como se hacía. Me ha gustado más de lo que esperaba, debido a que saber todos los conceptos sobre cómo crear un proyecto como éste puede llevar a representar agentes, por ejemplo, con inteligencia artificial que tengas en mente a implementar y ver su desarrollo si es correcto o no mediante una visualización mejor para toda persona que quiera ver tu proyecto al respecto.

Por otro lado, ha sido algo costoso el seguimiento debido a la situación en la que nos encontramos. Creo que, en mi opinión, debería de haber más comunicación entre las diferentes asignaturas para saber realmente qué carga de trabajo hay hacia sus alumnos a nivel general. Aún así, se agradece la flexibilidad que ha habido a la hora de atrasar días de entregas de prácticas cuando las hemos necesitado.

Por último, me gustaría haber tenido algo más de tiempo o menos presión respecto a ella para realizar muchas más funcionalidades al proyecto. Aunque es normal que al final haya que evaluar en un tiempo finito sobre ello. Lo bueno de un proyecto como el mío ha sido que cada vez que se implementa algo nuevo, aparecen una infinidad de nuevas formas o funcionalidades a añadir por lo que he intentando centrar en el tiempo las que he considerado más importantes.

BIBLIOGRAFÍA

- Asignación de colores a luces y materiales: https://community.khronos.org/t/color-tables/22518/5

- Información sobre cómo realizar el movimiento de la bola con teoremas físicos: http://laplace.us.es/wiki/index.php/Caso_de_movimiento_con_aceleración_constante_(GIE)#Velocidad_inicial_y_aceleraci.C3.B3n

ANEXO. MANUAL DE USUARIO

- **‘v/V’:** Cambiar la posición de la cámara.
- **‘a/A’:** Activar/Desactivar animación del grafo de escena (Clank).
- **‘b/B’:** Activar/Desactivar animación de la bola. Para comenzar a jugar, pausar el juego, reanudarlo.
- **‘q/Q’:** Activar el movimiento de la palanca izquierda.
- **‘w/W’:** Activar el movimiento de la palanca derecha.
- **‘+/-’:** Zoom in/out.
- **‘p/P’:** Cambiar el tipo de proyección (Paralela y Proyección).
- **‘27’:** Tecla de escape para SALIR.
- **“FLECHA HACIA ARRIBA”:** movimiento positivo de la posición de la cámara en el eje Z.
- **“FLECHA HACIA ABAJO”:** movimiento negativo de la posición de la cámara en el eje Z.
- **“FLECHA HACIA DERECHA”:** movimiento positivo de la posición de la cámara en el eje X.
- **“FLECHA HACIA IZQUIERDA”:** movimiento negativo de la posición de la cámara en el eje X.