Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Reporte de proyecto terminal

IMPLEMENTACIÓN DESDE CERO
DE UN MOTOR DE JUEGO DE ESTILO
”LABERINTO DE CANICAS”

Meza López Luis

Matrícula 205359472

Trimestre 11-O

Asesor Dr. Risto Rangel Kuoppa

1. Introducción:

La propuesta de proyecto terminal plantea el diseño e implementar desde cero un motor de juego1 3D2 y utilizar el motor para construir un juego de estilo “laberinto de canicas”.

En este reporte se describirá como se realizaron cada uno de los módulos principales: inicialización del juego, gestión de entradas, lógica de comportamiento, colisiones, movimientos de objetos, renderización y acciones post juego.

También se puntualizan los alcances de los objetivos particulares y objetivos generales establecidos, los cuales permiten generar una explicación detallada del desarrollo del videojuego.

2. Descripción del proyecto:

El **MOTOR DE JUEGO DE ESTILO”LABERINTO DE CANICAS”** funciona en tiempo real. Esto quiere decir que obtiene del sistema una respuesta en un tiempo tan rápido que el usuario tiene la impresión de que es prácticamente instantáneo. Esta característica es requerida para tener un juego interactivo.

Durante la ejecución de la aplicación, en el juego se realizan tareas como dibujar objetos, calcular posiciones, actualizar coordenadas de los personajes y detectar colisiones. Todo esto independientemente de si recibe eventos[1] por parte del usuario o no [3].

En el juego se pueden realizar diferentes acciones:

1.- Recorrer un mundo virtual diseñado como un laberinto de canicas

- Desplazándose por las superficie plana del laberinto utilizando el teclado como medio de entrada en la ejecución del juego.
- Girando en 360° sobre sus tres ejes(x, y, z) dentro del mundo virtual, lo cual permite cambiar el ángulo de vista para buscar el más adecuado, esto se realiza deslizando el mouse sobre una superficie.

2.-Interactuar con las gráficas del juego

- Graficas del menú. Siempre están visibles durante toda la ejecución del juego, lo cual permite que el jugador cambie el estado de esté durante todo el tiempo de ejecución

La información que se presenta en la gráfica, ver Figura1, del menú es la siguiente:

- Gráficas del menú:

[1]Evento se da cuando el usuario pulsa algún botón o tecla que permite ejecutar una función asociada a dicha acción.

- Jugar: representa el icono para iniciar el juego y desencadenar eventos en el tiempo de ejecución del mundo virtual.
- Ayuda: representa el icono que muestra las instrucciones para manejar el juego
- Información: Representa el icono donde se muestran los datos del juego.

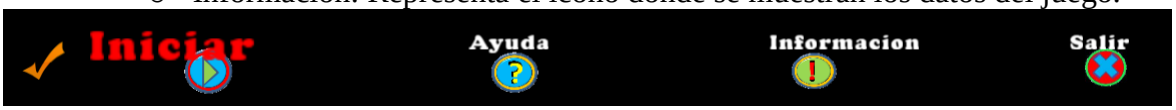


Figura1. Gráficas del menú

En las siguientes secciones se presentan las descripciones más detalladas de cada uno de los módulos que componen el ciclo de vida del videojuego.

3. Estructura del proyecto:

Inicialización de componentes

Para construir los elementos del juego se hicieron clases y métodos, heredan de una clase base llamada Game, la cual contienes los métodos y atributos que varían según el tipo de uso que tengan, ya sea para posicionar vectores o para cambiar direcciones. En la Figura2 se observa un diagrama general de lo antes mencionado.

En la clase Game se inicializan los componentes, con un *vector3*[2] de la cámara. La declaración de vértices son inicializados para hacer uso de ellos al momento de renderizar[3] y el tamaño de la pantalla se especifica en el constructor Game.

En el método Loadcontent() se detallan las rutas de los modelos, imágenes, texturas, y efectos de sonido que serán utilizadas durante la ejecución del juego. Las librerías que utilizamos están dadas automáticamente por el programa XNA.

[2] segmento de recta dirigido punto en el espacio de tres dimensiones con componentes en (x, y, z).

[3] Renderizado es un término usado en programas de diseño en 3D y aplicado en la computación gráfica para referirse al proceso de generar una imagen desde un modelo.

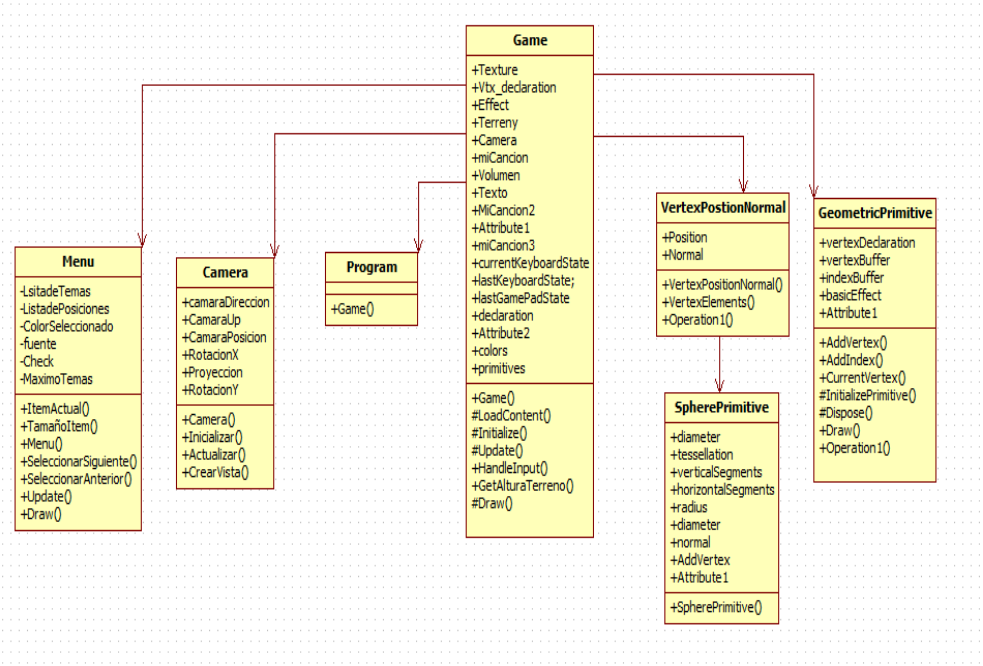


Figura2. Entidades para construcción del mundo virtual.

Gestión de Entradas

Para entender las acciones de este módulo hago uso del método de actualización en tiempo de ejecución, en este verifico las entradas del ratón y del teclado, primero actualizando las selecciones del menú, volumen del sonido, y de acuerdo a la selección se verifican los eventos que deben ser iniciados, de ser así, actualizamos la posición de la cámara para saber si se tiene que mover hacia una posición futura, desplazar o regresar a su estado inicial, lo que a esto llamo posición segura.

Las actualizaciones cambio de colores en la esfera ver Figura3, cambio de primitivas ver Figura4, velocidad de giro de la esfera (constate), son llamadas desde una función que he llamado handleInput. Las colisiones y la vista de la sombra de la canica también son utilizadas en este método.

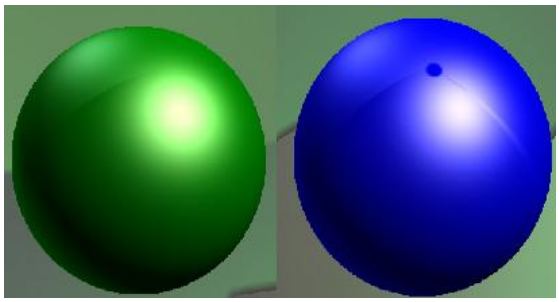


Figura3. Cambio de color en canica

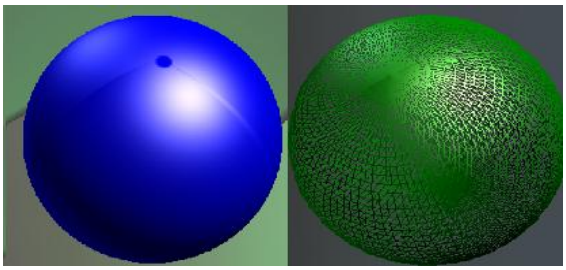


Figura4. Cambio de primitiva en modo de relleno solido a modo alambre en la canica

Lógica de comportamiento

Detección de colisiones entre la cámara y terreno (laberinto)

En la propuesta del proyecto se especificó que se utilizaría *octrees*[4] para calcular las colisiones entre la canica y las paredes del laberinto. No se usaron porque para la cantidad de elementos del mundo virtual era más costoso programarlos, que hacer simplemente una comparación exhaustiva. Por lo contrario encontré una posible solución moviendo directamente la cámara (en primera persona) sin "atravesar las paredes".

Con el terreno generado mediante un mapa de alturas el problema hay que afrontarlo de otro modo. No se puede utilizar la clase *BoundingSphere* [5]ni *BoundingBox* [6]por que la detección de colisiones sería enormemente imprecisa, Esta clase se usa solo para rodear el modelo de un objeto y en el laberinto tendría que rodear todas las paredes, sería complicado y extenso el cálculo para cada una. Por lo tanto esta solución es difícil que nos sirva para los terrenos.

Para hacer los cálculos más eficientes, lo que parece más sencillo es lo siguiente: en el momento de generar el terreno a partir del mapa de alturas, estamos creando vértices para nuestro nuevo modelo 3D. Almacené esa información de forma fácilmente accesible, por ejemplo en una lista de vértices. Cuando movamos la cámara sólo deberemos buscar en la lista de vértices, el valor de Y para el terreno, en relación a la posición Y del vector posición de la cámara.

Vemos la ilustración con un dibujo de la Figura5 para que se entienda mejor.

---

[4] Octree es una estructura de datos de tipo árbol en el cuál se tienen en cada nodo interno hasta ocho hijos. Se usa frecuentemente para repartir el espacio tridimensional en ocho octantes de forma recursiva.

[5] BoundingSphere son cuerpos geométricos en forma de esfera que no se renderizan, pero se utilizan para el cálculo de colisiones.

[6] Boundingbox son cuerpos geométricos en forma de cubo que no se renderizan, pero se utilizan para el cálculo de colisiones.

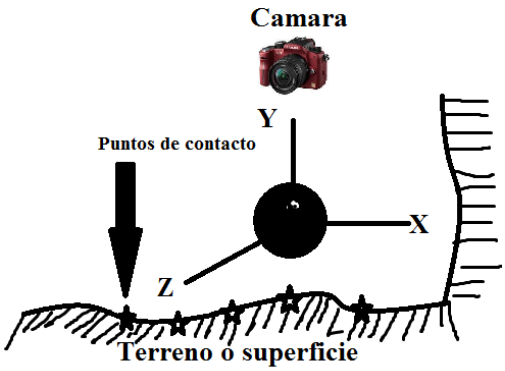


Figura5. Representacion de colisiones entre el terreno y la cámara

La idea es simplemente que al desplazar la cámara por cualquiera de sus ejes, se compruebe el elemento Y del vértice del terreno más cercano para la posición X-Z de la cámara. Si el usuario mueve la cámara a una posición en la que hay colisión, incrementamos la posición de la cámara para evitar la misma [5].

De este modo obtenemos la sensación de que la cámara "flota" en dirección de la canica desplazándose por encima del terreno generado, cuando hay colisión con un muro solo basta con cambiar la posición de la cámara y se crea el efecto de estar por encima del muro, esta acción se repite consecutivamente durante toda la actualización del tiempo juego. El resultado es el siguiente, ver Figura6:

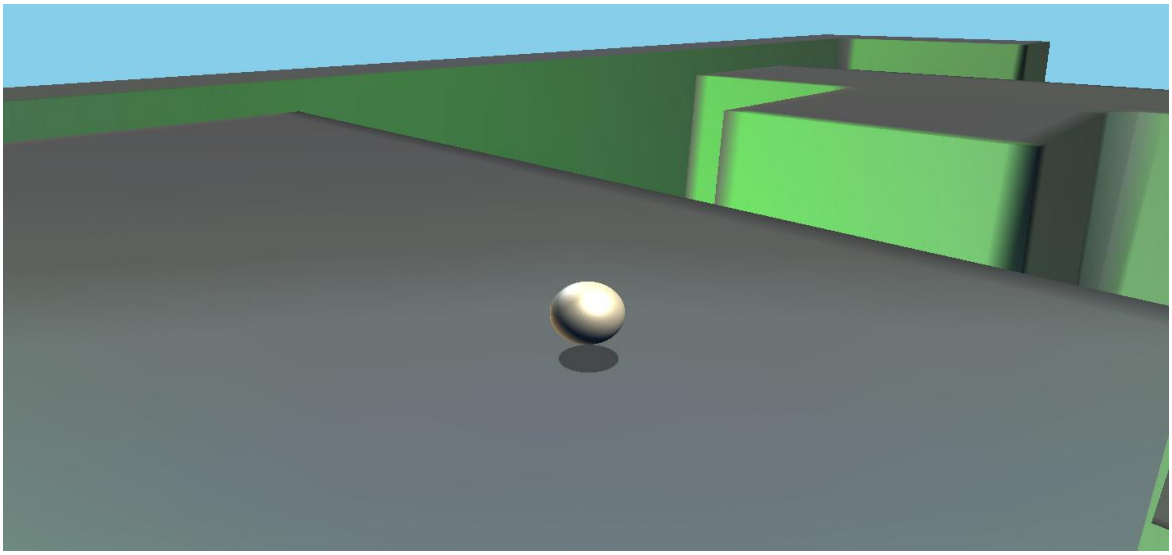


Figura6. Resultado de la colisión entre la canica y las paredes

Movimientos de objetos

Cámara

Para crear una vista de 360 grados se hicieron componentes reutilizables para aplicaciones 3D para ello se utilizaron vectores para posición y dirección. También las matrices de vista y proyección fueron fundamentales para que esta mirara en dirección apropiada.

La velocidad de rapidez se estableció fácilmente como un dato de tipo float llamada Speed, al igual que se usó una función llamada MathHelper, la cual permite calcular el ángulo máximo de los movimientos de la cámara y las rotaciones siempre fueron actualizadas en los vectores en X y en Y con limitaciones en los ángulos en Y.

Se utilizó un constructor para inicializar la vista de la cámara. El cual hace uso de un *vector3* de posición y se genera la proyección a partir de una matriz que usa un campo de la perspectiva de la visión.

La posición se estableció siempre en el centro de la pantalla y se dejó que la cámara siempre mirara en dirección de la matriz de vista que se creó.

Renderización

Generación de terreno del laberinto

Para la generación del laberinto se optó por el renderizado de terrenos a partir de mapas de altura que consiste en generar programáticamente el terreno más o menos montañoso o con paredes en su interior a partir de un *bitmap*[7] en blanco y negro [6].

El algoritmo que se usa está basado en leer todos los colores del *bitmap* y establecer una altura en un vector tridimensional a partir de los elementos que estén contenidos en la imagen del terreno y de su coloración. Posteriormente se añadirán los triángulos que conforman el terreno.

Lo importante de la generación del terreno se hace con una clase que extiende el Content Pipeline. De este modo, podemos tratar nuestro terreno como una textura más, convirtiéndola en un nuevo dato de tipo float para su fácil procesamiento, la cual se puede cargar, y la podemos generar con una sencilla invocación del método "LoadContent" que utiliza XNA, donde terreno es una variable de tipo "ProcessadorTerrenys". La imagen del terreno se diseña de tipo plano con laberintos en su interior que permiten visualizar una entrada y una salida véase Figura7.

[7] Bitmap es estructura de datos que representa una rejilla rectangular de píxeles o puntos de color

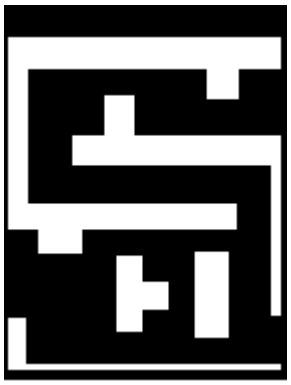


Figura7. Imagen de laberinto creada para generar terreno

Para darle texturas a las paredes se utilizó una imagen que está en formato *bmp*[8] con la cual se asoció como material a la textura del terreno tomándola como una referencia externa que se invoca desde el método LoadContent, Dónde Textura es una variable de tipo Texture-XNA Framework, ver Figura8.

Figura8. Imagen de rocas para generar textura en formato bmp tomada de la referencia [7]

Dibujar terreno

Una vez que se realizó el terreno debemos dibujarlo en la pantalla, para lo cual se usó una función llamada DibuxarTerreny dentro del método Draw de la clase Game donde se toma en cuenta las matrices de vista y proyección para agregar efectos de iluminación y de texturas al modelo del terreno.

En la Figura9 se puede observar el resultado final del laberinto en tres dimensiones.

[8] Bmp es la extensión de archivos de imagen del formato llamado Mapa de Bits

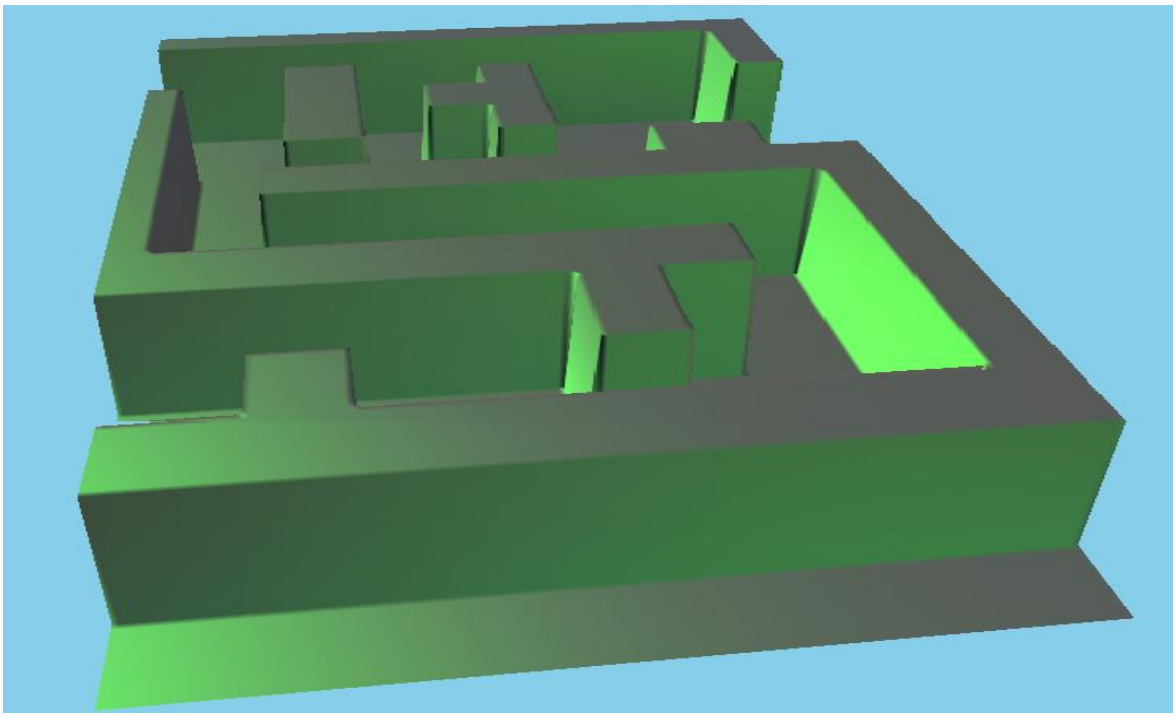


Figura9. Vista final del laberinto generado programáticamente

Dibujar sombra

Para la sombra de la canica se utilizó un sprite cargado como una textura a la cual se le asignó efectos básicos y por medio de vértices se define la posición y coordenadas de la textura para que aparente estar debajo de la canica.

En el método de actualización se condicionó que la sombra se dibuje si hay colisiones o se borre cuando no las hay, de esta manera podemos aparentar que la esfera toca el piso.

Dibujar canica

Para la representación gráfica de la canica en la pantalla se trabajó con *sprites*[9] estático en 3D con el fin de lograr que la canica se mueva por toda la pantalla y colisione, sin dejar de ser un *Sprite* estático.

Para crear la canica se utilizaron geometrías primitivas, la cual se define como una clase SpherePrimitive, con un tamaño definido y con un nivel de teselado. Se hicieron vértices en la parte superior e inferior con anillos en las latitudes, siendo estos cada vez más elevados a la mitad de la esfera.

[9] Sprite estático es un mapa de bits que se dibuja en la pantalla, en el cual el fondo de la imagen está en movimiento, mientras otra imagen permanece fija.

Se llenó el cuerpo de la canica con triángulos que une cada par de anillos de la latitud, después, se dibujó el modelo utilizando efectos básicos para las primitivas con sets de parámetros y estados de renderizacion, la Figura10. Muestra el resultado de la canica.

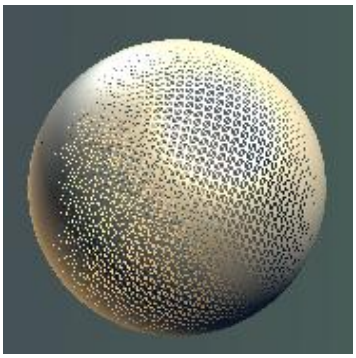


Figura10. Imagen de la canica

Una vez que se han implementado todos los módulos, hago uso de ellos para generar el mundo virtual y se actualiza el estado del juego para cumplir con el ciclo lógico, por lo tanto esta es una imagen de la vista final del juego, ver Figura11.

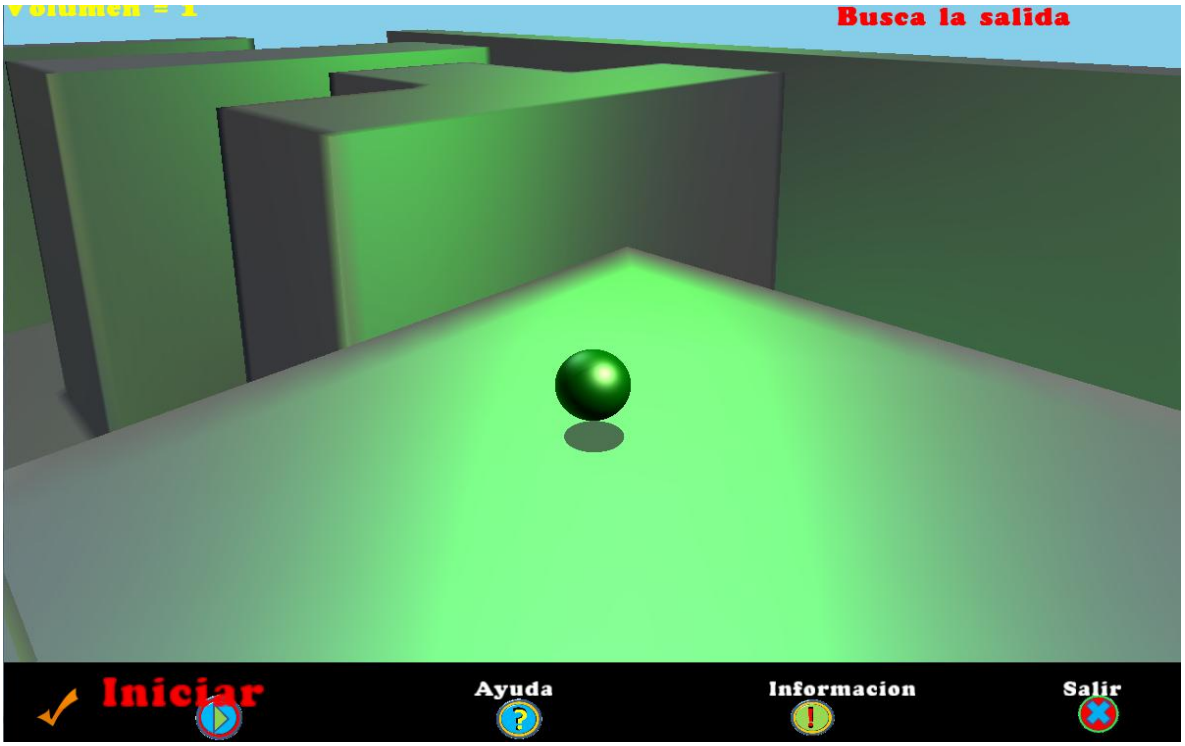


Figura11. Vista final del mundo virtual (Laberinto de canicas)

Acciones post-juego

Menú

La clase menú es muy impórtate porque permite desencadenar los primeros eventos que permiten empezar o terminar el juego, como son:

- Jugar: Una vez seleccionado este ítem, el modelo del laberinto, la canica, los sonidos y otros son puestos en acción, lo cual permite iniciar el juego.

- Ayuda: Este ítem interrumpe distintos efectos como son los sonidos y la visualización del laberinto. La posición de la canica vuelve a su posición segura para evitar conflictos, iniciando así la muestra de una imagen que contiene instrucciones acerca de la operación del juego.

- Información: Este ítem interrumpe distintos efectos como son los sonidos y la visualización del laberinto. La posición de la canica vuelve a su posición segura para evitar conflictos, iniciando así la muestra de una imagen que ya se ha cargado inicialmente y que contiene información acerca de la los requerimientos del juego.

- Salir: termina la aplicación.

Juego finalizado

En la siguiente pantalla se puede visualizar una imagen que muestra el mundo virtual terminado, con algunos.

4. Manual de usuario:

Manual de usuario
Del Videojuego
Laberinto
De
Canicas 3D

Dirigido al usuario final

Elaborado por: Meza López Luis

Como proyecto terminal de ingeniería en computación

Introducción

Este documento está elaborado para dar a conocer el funcionamiento del motor de juego 3D, de estilo "laberinto de canicas". En su contenido se abordan temas que ayudan a la comprensión de las diferentes funciones que el juego permite realizar al momento de su uso.

Para explicar el funcionamiento del motor de juego de laberinto de canicas se utilizaran pantallas con numeración en cada una de las imágenes y seguido se cuenta con una explicación de cada una.

Análisis y requerimientos del sistema

Para llevar a cabo el funcionamiento del juego es necesario instalar el siguiente software:

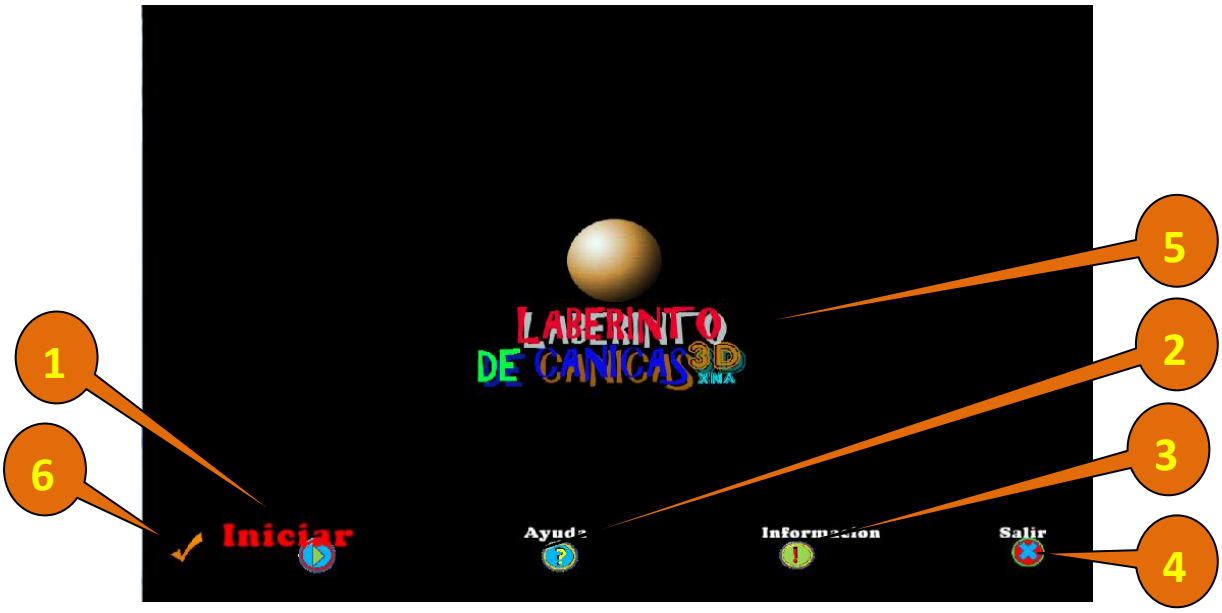
- Microsoft Visual Studio 2008 Express, descargado de: *Dreamspark* con licencia comercial de la siguiente liga:
  https://www.dreamspark.com/Products/Product.aspx?ProductId=29
- XNA Game Studio 3.1, descargado de *Dreamspark* con licencia comercial de la siguiente liga:
  https://www.dreamspark.com/Products/Product.aspx?ProductId=31

- Microsoft DirectX SDK, descargado de : softonic.com, con licencia comercial
  http://directx-9.softonic.com/

Una vez instalado el software especificado basta con ejecuta el icono con el nombre "Laberinto de canicas" y la siguiente pantalla será la que se muestra en la Pantalla1.

Explicación del funcionamiento

En la Pantalla1 se muestra la imagen principal, esta es la primera que aparece al momento de iniciar el juego, se aprecia un fondo oscuro y en la parte inferior se tiene un menú con cuatro temas, que permiten a l usuario interactuar con el juego.



Pantalla1. Pantalla principal del juego Laberinto de Canicas

1. Mena iniciar.
2. Menú de Ayuda.
3. Menú de información.
4. Menú Salir.
5. Logotipo del juego de Laberinto de Canicas.
6. Marca (flecha) que permite saber la selección del menú.

A continuación se definen los diferentes temas del menú.

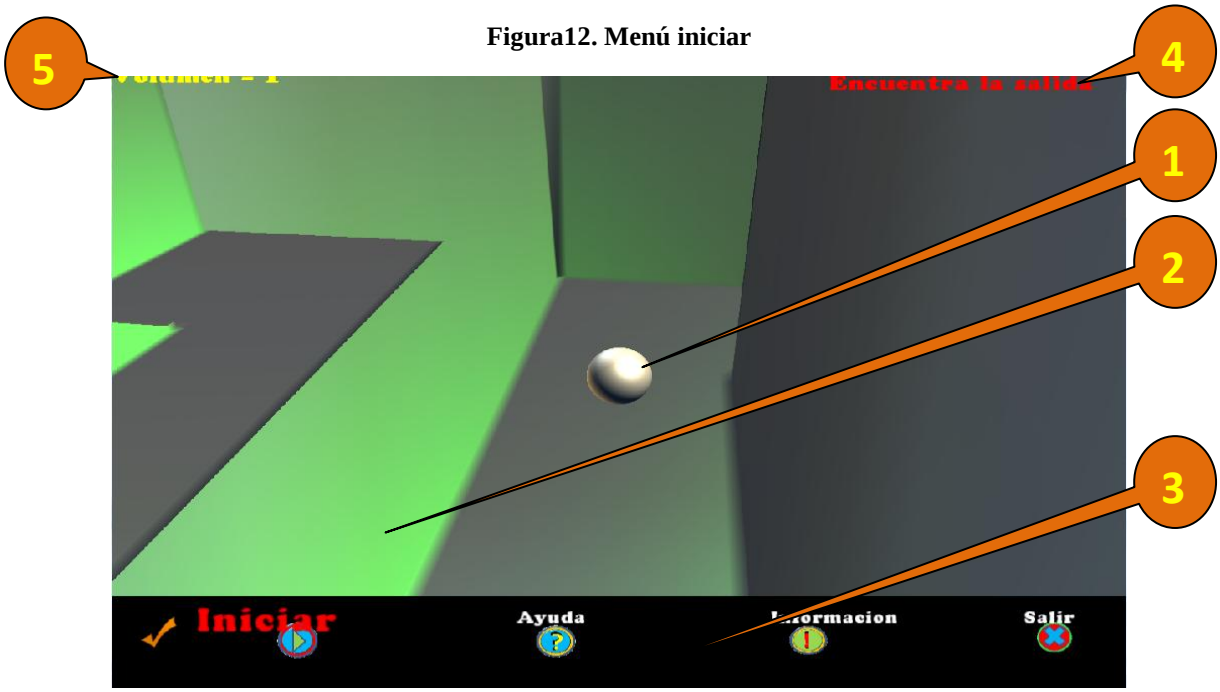
Menú Principal

Se cuenta con cuatro temas dentro del menú, los cuales se pueden seleccionar por medio de una paloma, la cual marca el tema a un costado izquierdo, para desplazarla por el menú solo basta con utilizar las teclas de navegación a la izquierda ← y a la derecha →.

Una vez iniciado el juego también puede hacerse uso del menú, ya que este esta disponible todo el tiempo de ejecución.

> **Menú Iniciar:** Permite comenzar el juego, se debe tener en cuenta que al seleccionar este tema el jugador debe estar listo para recorrer los distintos laberintos haciendo uso de la canica, el icono del tema seleccionado se ve en la Figura12. La selección de este tema permite visualizar la pantalla2.

Figura12. Menú iniciar

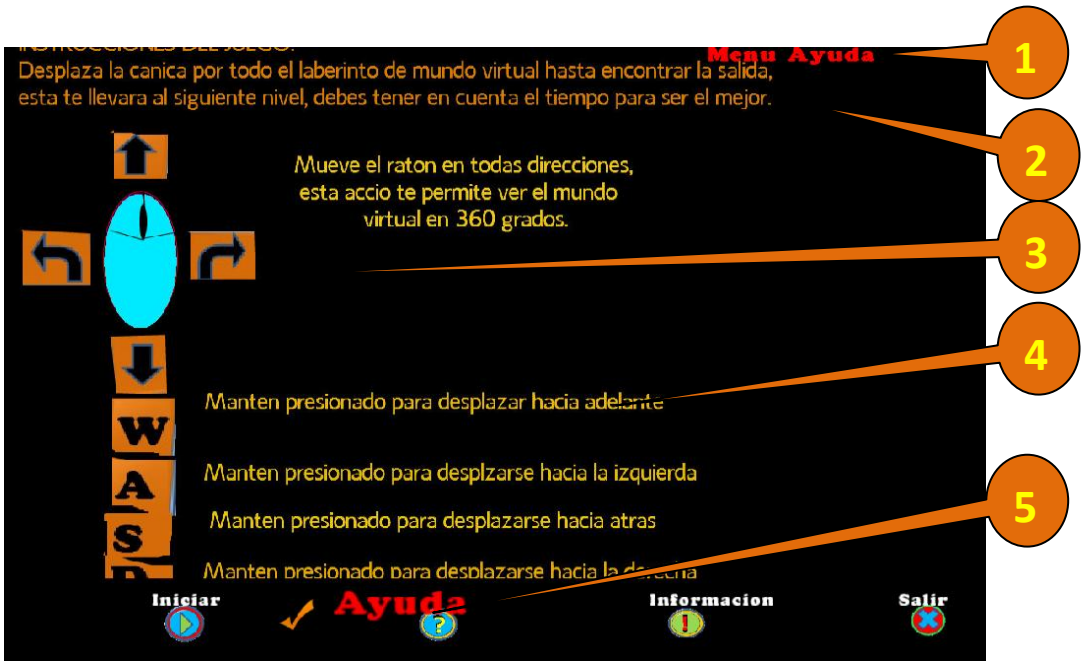


Pantalla2. Posición inicial de la canica al momento de seleccionar el tema iniciar del menú

1. Canica: es la imagen principal, con ella puedes avanzar dentro adentro del laberinto.
2. Muros: son las paredes que limitan el paso de la canica, el conjunto de muros hace que el laberinto se forme, recuerden que la canica debe de chocar lo menos posible con ellos.
3. Menú. El menú siempre está disponible para que el usuario pueda hace uso de esté, seleccionado el tema que quiera, presionado las teclas de dirección.
4. Mensaje encuentra la salida: en la parte superior derecha de la pantalla siempre estará visible algún mensaje que indica las acciones que suceden en el juego, para este caso, el mensaje avisa que al iniciar el juego, se debe buscar la salida del laberinto.
5. Volumen: este mensaje que se encuentra en la parte superior derecha define el nivel de volumen, se limita a un nivel máximo de sonido con valor en "1" haciendo uso de la tecla ↑ de navegación para elevarlo y el mínimo de sonido con valor de "0" haciendo uso de la tecla de navegación ↓. para bajar el volumen.

Menú ayuda: Permite visualizar al pantalla3. Aquí se encuentran las instrucciones y objetivo, se define el uso de cada una de las teclas que se permite utilizar en el juego, el icono del tema seleccionado se ve en la Figura13.

Figura13. Menú ayuda



Pantalla3. Instrucciones de ayuda

1. Mensaje de selección Ayuda: en la parte superior derecha de la pantalla siempre estará visible algún mensaje que indica las acciones que suceden en el juego, para este caso, el mensaje avisa que se encuentra en la pantalla de ayuda.

2. Instrucciones acerca del objetivo del juego.

3. Imagen del ratón, el cual especifica físicamente que tiene puede moverse en distintas direcciones. Una especificación es que se puede mover en 360°.

4. Descripción de la utilización del teclado, par este caso solo podemos hacer uso de la tecla:

    - W: mantener presionada para desplazarse hacia adelante y hacer que la canica y flote.
    - A: mantener presionada para desplazarse hacia la izquierda.
    - S: mantener presionada para desplazarse hacia atrás.
    - D: mantener presionada para desplazarse hacia la derecha.

5. Selección actual del menú ayuda

Menú información: Permite visualizar al pantalla4. Aquí se encuentran las instrucciones y objetivo, se define el uso de cada una de las teclas que se permite utilizar en el juego, el icono del tema seleccionado se ve en la Figura14.

Figura14. Menú información

Panatalla4. Información del juego

1. Mensaje de selección Información: en la parte superior derecha de la pantalla siempre estará visible algún mensaje que indica las acciones que suceden en el juego, para este caso, el mensaje avisa que se encuentra en la pantalla de información.

2. Información del nombre del motor de juego.

3. Selección actual del menú Información.

Menú salir: Permite terminar la aplicación, este tema cierra todas las ventanas. El icono del tema seleccionado se ve en la Figura15.

Figura15. Menú salir

Iniciando el Juego

Una vez iniciado el juego la canica se muestra en un punto inicial, dentro del espacio del laberinto formado por las paredes. El jugador deberá de utilizar las teclas antes mencionadas para desplazar la canica sobre el mundo virtual y girando el ratón en distintas direcciones, para buscar las mejores posiciones de visibilidad.

Al inicio del juego debe escucharse un sonido de música de fondo, esta música indica que el juego ha iniciado. Si la canica avanza y tiene contacto con el piso también se escucha un efecto de choque la cual indica que estas desplazándote.

El jugador debe recorrer todo el mundo virtual y evitar los choques con las paredes y encontrar las posibles salidas del laberinto.

Dificultad del juego

Una vez que se haya iniciado el juego deberás tener en cuenta que existe una dificultad para pasar los laberintos y deberás desarrollar una habilidad para entender la complejidad de las paredes y en ciertos casos, no podrás pasar por encima de sombras ya que estas te llevaran a un punto alto, el cual cambia el modo de vista del juego, de un punto bajo a uno más alto y así desplazarte seguir buscando la salida.

5. Manual de programador:

Introducción

En este documento vamos a describir los diagramas que nos sirven para implementar todo el motor de juego de tipo laberinto de canicas, primero los diagramas de entidad-relación y después el diagrama de estados, seguido del diagrama de flujo de información para cada una de los módulos del estado del juego.

Para la programación del juego y la conexión de módulos utilicé *C#*[10]. Con XNA versión 3.1 se manejaron las cuestiones gráficas con el Microsoft DirectX SDK. Ambos fueron instalados en el sistema operativo Windows Vista[11].

Entornos de desarrollo y API gráfica:

- Microsoft Visual Studio 2010 Express, descargado de: *Dreamspark* con licencia comercial de la siguiente liga:
  https://www.dreamspark.com/Products/Product.aspx?ProductId=29
- XNA Game Studio 3.1, descargado de *Dreamspark* con licencia comercial de la siguiente liga:
  https://www.dreamspark.com/Products/Product.aspx?ProductId=31

Diagrama de entidad relación

En el siguiente diagrama de la Figura16, se muestran las diferentes entidades y su relación entre ellas basado en una percepción del mundo real. Para modelar los datos del juego el modelo expresa entidades relevantes para el juego, así como sus interrelaciones y propiedades [8].

Se cuenta con seis entidades, cada una tiene atributos, aunque en la tabla1, solo se hace énfasis en las más importantes. También hay relaciones entre ellas, de igual manera solo se especifican las de mayor jerarquía, aquí una lista con algunas de ellas:

Tabla 1. Entidades y atributos del juego laberinto de canicas.

Entidad: Game	**Entidad:** Camera	**Entidad:** Geometric Primitive	**Entidad:** SpherePrimitive	**Entidad:** Menu	**Entidad:** Vertex Positio Normal
Atributos	**Atributos**	**Atributos**	**Atributos**	**Atributos**	**Atributos**

[10] C#: Lenguaje de programación orientado a objetos
[11] Windows vista es el nombre de una serie de sistemas operativos desarrollados por Microsoft

Cámara	Projection	Indices	Diametro	ItemsMenu	Posición
Modelo	View	Vertices	Radio	Checks	Normal
Textura	CameraUp	BasicEffect	Tessellation	Seleccionado	
…	…	…	…	…	…

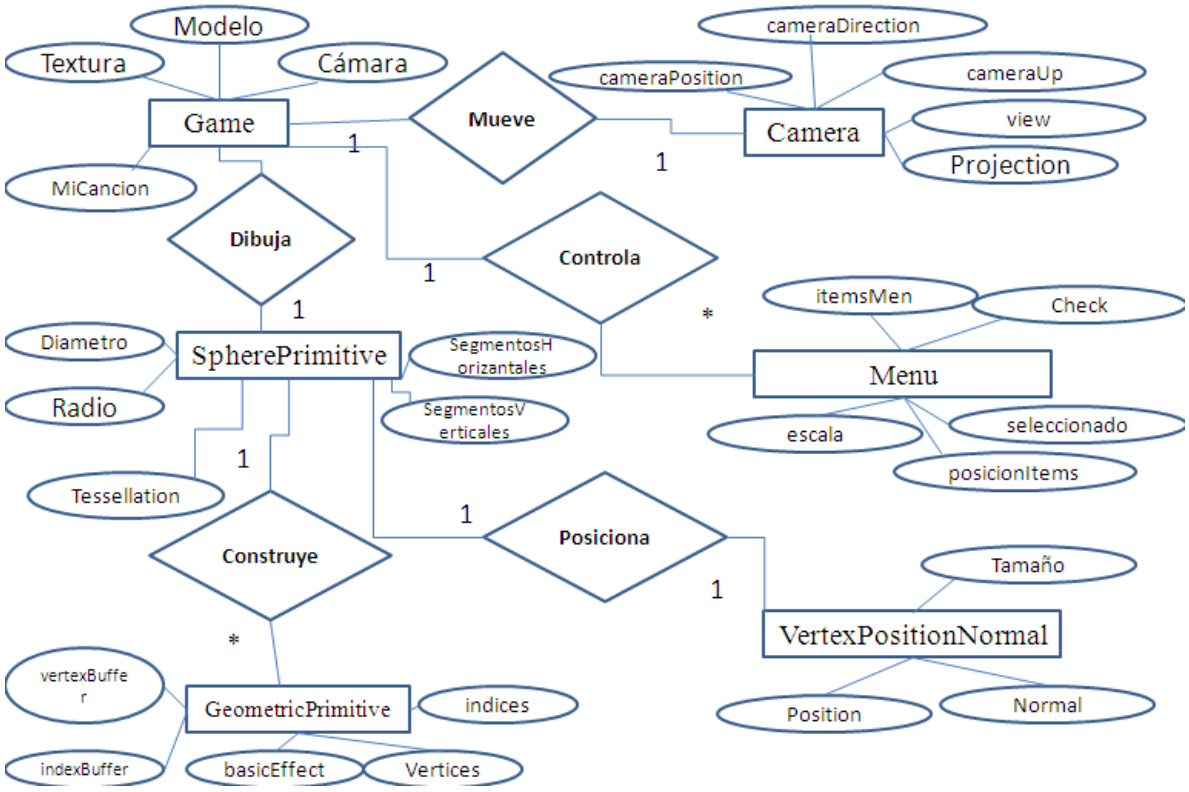


Figura16. Diagrama de entidad relación para el motor de juego laberinto de canicas

Diagrama de estado

En el diseño de videojuegos es importante tener en cuenta los diferentes estados en los que un juego puede encontrarse, y que dependen de las acciones que el usuario o jugador sigan.

El diagrama de estados diseñado para Laberinto de canicas que se muestra en la Figura17. Los círculos representan los diferentes estados que el juego puede tener y las flechas las condiciones que se tienen que dar (resumidas) para pasar de un estado a otro

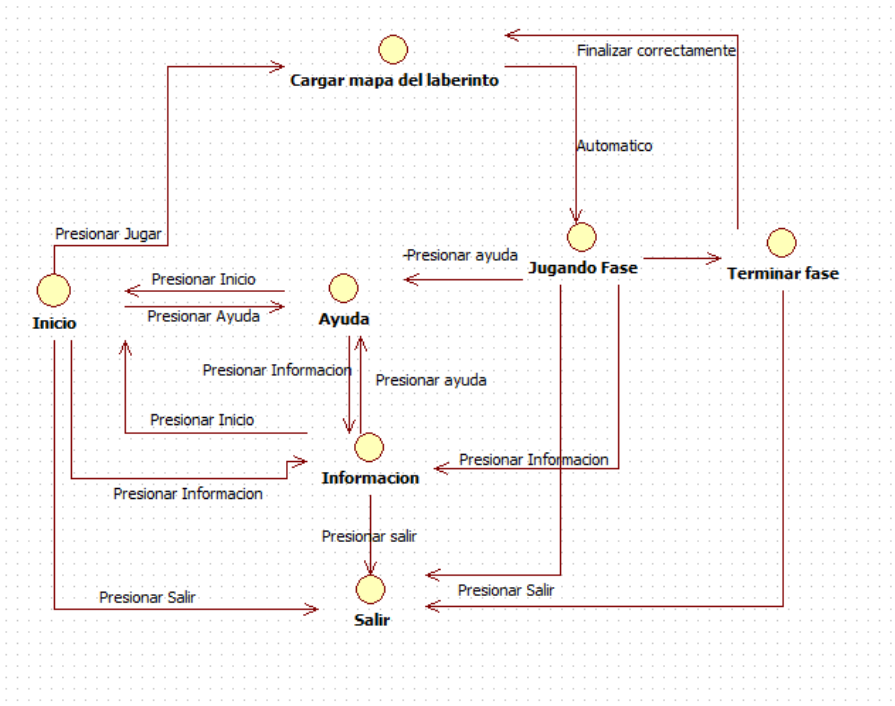


Figura17. Diagrama de estados del juego laberinto de canicas

El diagrama de estado tiene inconcluso algunos estados como cambiar de fase para el nivel al terminar el recorrido en el laberinto y al terminar el tiempo de juego se deberá pasar del estado jugando al estado salir.

Diagrama de flujo

El diagrama de flujo que se muestra en la Figura18, es la representación del comportamiento de la información que hay en el juego, cada módulo consta de varias etapas que pueden suceder en diferente orden.

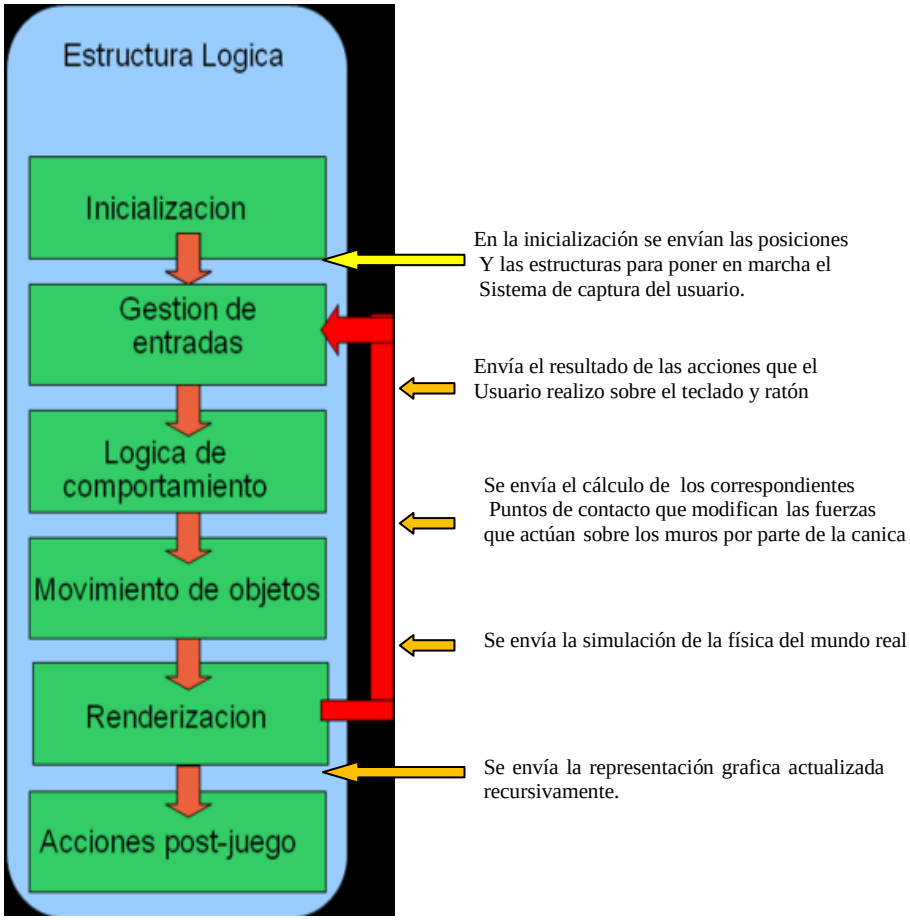


Figura18. Diagrama de flujo del juego

Cada uno de los bloques del juego realiza las siguientes funciones:

- **La inicialización del juego:** Es un módulo fundamental del juego. Es primordial que todo lo que sea necesario durante el juego esté inicializado desde antes que sea utilizado. Como mínimo inicializar la ejecución del subsistema de video que permite mostrar imágenes en la pantalla.

- **Gestión de Entradas:** En este módulo se obtienen las acciones que ha realizado el usuario sobre los dispositivos de entrada como el teclado y el ratón, se sabrá, si se ha movido el ratón, las posiciones de origen o destino y las teclas que se han presionado.

- **Lógica de comportamiento:** El laberinto de canicas es meramente reactivo a la física simulada. Por eso que en este módulo se detectan las colisiones de la canica con los muros y se calculan los correspondientes puntos de contacto que modifican las fuerzas que actúan sobre ella.

- **Movimientos de objetos:** En este módulo, implementamos el motor de física.

- **Renderización:** En este módulo se dibujan en la pantalla las imágenes y objetos durante el ciclo lógico del juego.

- **Acciones post-juego:** En este módulo se comprueba el estado del juego para procesa su finalización o la existencia de un error.

6. Conclusiones:

De acuerdo a los objetivos planteados al inicio del proyecto afirmo que se lograron alcanzar. En cuanto a su funcionalidad, se lograron conjuntar todos los módulos especificados en la propuesta, aquí señalo algunos de los más importantes:

- El modulo de gestión de entradas funciona correctamente y atiende a tiempo las ordenes desde la inicialización hasta las acciones post-juego

- El modulo de movimientos de objetos se calcula y funciona acorde a las restricciones de simulación, con velocidad (sin fricción y sin inercia).

- La canica responde al cálculo de colisiones con el piso y con las paredes, dando respuesta a los eventos de choque.

- La renderización despliega las pantallas correctamente durante todo el proceso de juego.

- El Motor de juego que simula un terreno con paredes igual a un laberinto está terminado y puede desplazarse la canica libremente, con visibilidad en tres dimensiones y visualizar una posible salida.

7. Bibliografía:

[1] Artículo de Gonzalo A. Ramón sobre inteligencia lúdica y laberintos
http://www.uned.es/inteligencialudica/Documentos/Laberinto.pdf
Consultada 16 de Octubre del 2010

[2] Inteligencia lúdica y laberintos
http://www.uned.es/inteligencialudica/Documentos/Laberinto.pdf
Consulta: 16 de Octubre del 2010

[3] Soluciones informáticas para la visualización tridimensional en tiempo real
http://www.cecam.sld.cu/pages/rcim/revista_5/articulos_htm/info rmatica.htm#4
Consulta: 08 de Noviembre del 2010

[4] Utilización de octrees
http://es.wikipedia.org/wiki/Octree
Consulta: 08 de Septiembre del 2011

[5] Colisiones entre camara y terreno
http://geeks.ms/blogs/jbosch/archive/2009/07/16/xna-detecci-243-n-de-colisiones-entre-una-c-225-mara-y-un-terreno.aspx
Consultada 16 de Octubre del 2011

[6] Generación de terrenos a partir de un bitmap
http://geeks.ms/blogs/jbosch/archive/2010/03/03/xna-game-engines-para-xna.aspx
Consultada 02 de Noviembre del 2011

[7] Imagen de rocas
http://jbosch.wordpress.com/2009/06/06/xna-generacio-terrenys-mapa-altures/
Consultada 25 de Noviembre del 2011

[8]Diagrama entidad relación
http://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n
Consultada 06 de Diciembre del 2011

9. Anexos:

Código fuente de las clases

Clase: Game

Proporciona la inicialización de dispositivos gráficos la lógica del juego y el código de procesamiento.

Funciones públicas miembro

flotar	**GetAlturaTerreno** (float x, float z, List <Vector3> vértices)

Funciones de miembro protegido

override void	**LoadContent** ()
override void	**Inicializar** () */// Permite el juego realizar cualquier inicialización que se necesite antes de empezar a*
override void	**Actualización** (tiempo de juego GameTime) *///En este metodo es en donde se ejecuta la l gica del juego, ya sea que puede*
override void	**Dibujar** (tiempo de juego GameTime) */// este método se dibujan las texturas, sprites, tienen 3 metodos principales, como son el begin, draw y end, entre otros*

Documentación de las funciones miembro

override void laberinto:: Juegos:: Initialize ( ) [En línea, protegido]

Permite el juego realizar cualquier inicialización que se necesite antes de empezar a ejecutarse. Esto es donde puede consultar para cualquier servicio necesario y cargar / cualquier contenido relacionado que no sea gráfico. La llamada a base. Initialize enumera todos los componentes y los inicializa.

override void laberinto:: Juegos:: Actualización **(GameTime** ***tiempo de juego*** **) [En línea, protegido]**

///En este método es en donde se ejecuta la lógica del juego, ya sea que puede ser colisiones, actualizaciones del mundo, audio, etc ///

La documentación para esta clase fue generada a partir del siguiente archivo:

- Game.cs

```
#region Bibliotecas
//Librerias de XNA que se utilizaran posteriormente
using System;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Storage;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Media;
using Primitives3D;
using MenuXNA;
#endregion

namespace laberinto
{
   /// <summary> /// Clase: Game
   /// proporciona la inicialización de dispositivos gráficos
   /// la lógica del juego y el código de procesamiento.
   /// </summary>
   ///
     public class Game : Microsoft.Xna.Framework.Game
   {

         GraphicsDeviceManager graphics;
         SpriteBatch spriteBatch;

         #region Variables Sombra
         Texture2D texture;
         VertexDeclaration vtx_declaration;
         int[] indices = new int[6];
         VertexPositionTexture[] vtxs = new VertexPositionTexture[4];
         BasicEffect effect;
         bool enable_bg = false;
         float alpha = 0.7f;
```

```
#endregion

#region Variables del Terreno
Model terreny;
Camera camera;
const float margenSeparacionTerrenoCamara = 200;
#endregion

#region Varibles Musica
Song miCancion;
float incVolumen = 0.005f;
SpriteFont texto;
String Volumen;
private SoundEffect miCancion2,miCancion3;
#endregion

#region Variables Menu
SpriteFont mouse;//sprite del seleccion
Menu mimenu; // clase del Menú
//Int32 mx, my; // posiciones del puntero del mouse
String seleccionado;
//bool mpressed, prev_mpressed = false; // variables usadas en el control del botón del Mouse
bool menuUp, menuDown = false; // variables usadas en el control de las teclas
#endregion

#region Variables Fondo de Menu
Texture2D BotonPlay,BotonAyuda, BotonSalir,BotonInfo, Titulo,Ayuda,Info,Barra;
Vector2 FondoPos, CanicaPos, TituloPos, AyudaPos, InfoPos;
Rectangle Rectangulo;
#endregion

#region VARIABLES ESFERA

KeyboardState currentKeyboardState;
KeyboardState lastKeyboardState;
GamePadState currentGamePadState;
GamePadState lastGamePadState;
      VertexDeclaration declaration;
//VertexPositionColor[] vertices;
#endregion

#region LISTA DE COLORES PARA ESFERA
// Guarde una lista de colores tinte, además de que uno ya está seleccionado.
List<Color> colors = new List<Color>
 {
Color.BlanchedAlmond,
Color.Green,
Color.Blue,
 };
#endregion

#region Lista de Primitivas para esfera
List<GeometricPrimitive> primitives = new List<GeometricPrimitive>();
int currentPrimitiveIndex = 0;
int currentColorIndex = 0;
```

```
        //int colorfondo = 0;
        // ¿Estamos en el modo de representación alámbrica?
        bool isWireframe;
        #endregion

        #region Game
        // Constructor
        public Game()
        {
            //GraphicsDeviceManager maneja la configuración del dispositivo gráfico.
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";

            // Establesco el tamaño de la pantalla
            graphics.PreferredBackBufferWidth = 1200;
            graphics.PreferredBackBufferHeight = 850;
        }
        #endregion

        #region cargar contenidos
        // Metodo que utilizo para cargar  continuamente el contenido del pipeline
        protected override void LoadContent()
        {
            //creamos un nuevo SpriteBatch, lo podemos usar para dibujar texturas
            spriteBatch = new SpriteBatch(GraphicsDevice);
            //agrego primtivas de esfera
            primitives.Add(new SpherePrimitive(GraphicsDevice));
            // Cargamos el mapa del terreno "terreny"
            terreny = Content.Load<Model>("Laberinto1");
           // cargamos la textura para la sombra
            texture = Content.Load<Texture2D>("test");
            //inicializamos las instancias para la clase de efectos basicos
            effect = new BasicEffect(GraphicsDevice,null);
            //Vertices de posicion, texturas  para la sombra de la esfera
            vtx_declaration = new VertexDeclaration(GraphicsDevice,
VertexPositionTexture.VertexElements);
            vtxs[0].Position = new Vector3(-0.00f, -0.0f, 0.0f);
            vtxs[0].TextureCoordinate = new Vector2(0.0f, 0.0f);
            vtxs[1].Position = new Vector3(0.0f, -0.0f, 0.0f);
            vtxs[1].TextureCoordinate = new Vector2(.0f, 0.0f);
            vtxs[2].Position = new Vector3(0.0f, 0.0f, 0.0f);
            vtxs[2].TextureCoordinate = new Vector2(1.0f, 0.0f);
            vtxs[3].Position = new Vector3(-0.0f, 0.0f, 0.0f);
            vtxs[3].TextureCoordinate = new Vector2(0.0f, 0.0f);
            indices[0] = 1;
            indices[1] = 3;
            indices[2] = 1;
            indices[3] = 3;
            indices[4] = 2;
            indices[5] = 1;



            //carga contenidos de Musica
            miCancion = Content.Load<Song>("Musica");
```

```
            miCancion2 = Content.Load<SoundEffect>("EfectoCanica");
            miCancion3 = Content.Load<SoundEffect>("EfectoSeleccion");
            texto = Content.Load<SpriteFont>("Courier New");


            mouse = Content.Load<SpriteFont>("sp");//cargola fuente del las letras del menu
            mimenu = new Menu(Color.White, Color.Red, Content.Load<SpriteFont>("sp"),
Content.Load<Texture2D>("Check"));
            mimenu.addItemMenu("Iniciar", new Vector2(80, 750));//posicion de los menus en pantalla
            mimenu.addItemMenu("Ayuda", new Vector2(380, 750));
            mimenu.addItemMenu("Informacion", new Vector2(680, 750));
            mimenu.addItemMenu("Salir", new Vector2(980, 750));

            //Cargo los sprites del menu que se usa para botones y barras de menu
            Barra = Content.Load<Texture2D>("barra");
            Ayuda =Content.Load<Texture2D>("Ayuda");
            Info = Content.Load<Texture2D>("Informacion");
            BotonPlay = Content.Load<Texture2D>("Boton1");
            BotonSalir = Content.Load<Texture2D>("boton2");
            BotonAyuda  = Content.Load<Texture2D>("Boton3");
            BotonInfo = Content.Load<Texture2D>("Boton4");
            Titulo = Content.Load<Texture2D>("titulolab");

            //Asigno posiciones en pantalla sobre Vectores2 a los Sprites
            AyudaPos = new Vector2(800, 800);
            TituloPos = new Vector2(0, 0);//Posicion del Sprite "Laberinto de Canicas"
            FondoPos = new Vector2(0, 450);//Posicion del Sprite Play
            CanicaPos = new Vector2(0, 350);//Posicion del Sprite Cerrar
            InfoPos = new Vector2(800, 800);
            Rectangulo = new Rectangle(800, 100, graphics.GraphicsDevice.Viewport.Width,
graphics.GraphicsDevice.Viewport.Height);

          }
        #endregion

        #region Inicialitzación
        ///  <summary> /// Permite el juego realizar cualquier inicialización que se necesite antes de
empezar a
       /// ejecutarse.Esto es donde puede consultar para cualquier servicio necesario y cargar
       /// /// cualquier contenido relacionado que no sea gráfico.
       /// La llamada a base.Initialize enumera todos los componentes y los inicializa.
       ///  </summary>
        protected override void Initialize()
        {
            IsMouseVisible = false;//desparecemos el mause dentro del la pantalla
            seleccionado = "";//El string seleccionado debara estar sin ningun texto al iniciar

            //Inicializamos la camara con el vector3()en la siguiente posocion de inicio
            camera = new Camera(this, new Vector3(10, 20, 35),
                Vector3.Zero, Vector3.Up  );
            //agragamos componentes a la camara
            Components.Add(camera);

            //Sin vertexDeclaration no se puede renderizar
            declaration = new VertexDeclaration(GraphicsDevice, VertexPositionColor.VertexElements);
```

```
    GraphicsDevice.VertexDeclaration = declaration;

    base.Initialize();
  }
#endregion


 #region Actualizar la logica del juego
  ///  <summary> ///En este metodo es en donde se ejecuta la lógica del juego, ya sea que puede
                ///ser colisiones,actualizaciones del mundo,audio, etc
   ///  <summary> ///
  protected override void Update(GameTime gameTime)
  {

    //actualizacion del sonido y volumen  al presionar las teclas de arriba y abajo
   if (Keyboard.GetState().IsKeyDown(Keys.Up))
    {
       MediaPlayer.Volume += incVolumen;
    }
    else if (Keyboard.GetState().IsKeyDown(Keys.Down))
    {
       MediaPlayer.Volume -= incVolumen;
    }

   //texto en Panatalla de el nivel del volumne
    Volumen = "Volumen = " + MediaPlayer.Volume.ToString();

    //mando llamar la actualizacion del menu
    mimenu.Update(gameTime);


    //Actualizacion del Menu de selecciones
  KeyboardState k = Keyboard.GetState();
  if (k.IsKeyDown(Keys.Left) && menuUp)//si presiona tecla de flecha izquierda y menu arriba
  {
    mimenu.seleccionarAnterior();
    menuUp = false;
  }
  else if (k.IsKeyUp(Keys.Left))
  {
    menuUp = true;
  }
  if (k.IsKeyDown(Keys.Right) && menuDown)//si presiona tecla de flecha derecha y menu arriba
  {
    mimenu.seleccionarSiguiente();
    menuDown = false;
  }
  else if (k.IsKeyUp(Keys.Right))
  {
    menuDown = true;
  }
  if (k.IsKeyDown(Keys.Enter))
  {

    switch (mimenu.ItemActual)
```

```
            {
                case 0:

                    seleccionado = "Encuentra la salida";
                    FondoPos = new Vector2(-100, 0);//cambio la posicion del fondo,titulo
                    CanicaPos = new Vector2(-200, 0);
                    TituloPos = new Vector2(-500, -1000);

                    MediaPlayer.Play(miCancion);
                    MediaPlayer.IsRepeating = true;
                    AyudaPos = new Vector2(800, 800);
                    InfoPos = new Vector2(800, 800);
                    miCancion3.Play(10.10f);

                    break;
                case 1:
                    seleccionado = "Menu Ayuda";
                    AyudaPos = new Vector2(0,0);
                    TituloPos = new Vector2(-500, -1000);
                    InfoPos = new Vector2(800, 800);
                    MediaPlayer.Pause();
                    miCancion3.Play(10.10f);

                    break;
                case 2:
                    seleccionado = "Menu Informacion";
                    MediaPlayer.Pause();
                    miCancion3.Play(10.10f);
                    InfoPos = new Vector2(0,0);

                    break;
                case 3:
                    miCancion3.Play(10.10f);
                    this.Exit();
                    break;
            }

        }



            List<Vector3> vertices = (List<Vector3>)terreny.Tag;
            float alturaTerreno = GetAlturaTerreno(camera.cameraPosition.X, camera.cameraPosition.Z,
vertices);


            Window.Title = "Altura terreno: " + alturaTerreno + " Altura camara: " +
camera.cameraPosition.Y.ToString();

            // ------------------------------------------------
            if (Keyboard.GetState().GetPressedKeys().Length > 0)
            {
                Vector3 posicionFutura = new Vector3();
```

```
            // Miramos si hay que mover la cámara
            if (Keyboard.GetState().IsKeyDown(Keys.W))
            {
               posicionFutura = camera.cameraPosition + camera.cameraDirection * camera.speed;
            }
            if (Keyboard.GetState().IsKeyDown(Keys.S))
            {
               posicionFutura = camera.cameraPosition - camera.cameraDirection * camera.speed;
            }
            if (Keyboard.GetState().IsKeyDown(Keys.A))
            {
               posicionFutura = camera.cameraPosition + Vector3.Cross(camera.cameraUp,
camera.cameraDirection) * camera.speed;
            }
            if (Keyboard.GetState().IsKeyDown(Keys.D))
            {
               posicionFutura = camera.cameraPosition - Vector3.Cross(camera.cameraUp,
camera.cameraDirection) * camera.speed;
            }

            // Movemos la cámara
            camera.cameraPosition = posicionFutura;

            // Si nos hemos pasado moviéndonos hacia abajo, volvemos a la posición "segura"
            if (posicionFutura.Y - margenSeparacionTerrenoCamara < alturaTerreno)
            {
               camera.cameraPosition.Y = alturaTerreno + margenSeparacionTerrenoCamara;
               //si hay colision cambio de color la esfera
               currentColorIndex = (currentColorIndex + 1) % colors.Count;
               //si hay colision con el piso cambio a
               isWireframe = !isWireframe;
               miCancion2.Play(10.10f);

               //si hay colision entre la canica y el piso se dibuja la sombra
               vtxs[0].Position = new Vector3(-0.5f, -0.5f, 0.0f);
               vtxs[0].TextureCoordinate = new Vector2(0.0f, 1.0f);
               vtxs[1].Position = new Vector3(0.5f, -0.5f, 0.0f);
               vtxs[1].TextureCoordinate = new Vector2(1.0f, 1.0f);
               vtxs[2].Position = new Vector3(0.5f, 0.5f, 0.0f);
               vtxs[2].TextureCoordinate = new Vector2(1.0f, 0.0f);
               vtxs[3].Position = new Vector3(-0.5f, 0.5f, 0.0f);
               vtxs[3].TextureCoordinate = new Vector2(0.0f, 0.0f);
               indices[0] = 0;
               indices[1] = 3;
               indices[2] = 1;
               indices[3] = 3;
               indices[4] = 2;
               indices[5] = 1;
               seleccionado = "Busca la salida";


            }
            else//si no hay contacto entre la canica y el piso, no hay dibujo de la sombra y la posicion se
cambia
```

```
                {

                    vtxs[0].Position = new Vector3(-0.00f, -0.0f, 0.0f);
                    vtxs[0].TextureCoordinate = new Vector2(0.0f, 0.0f);
                    vtxs[1].Position = new Vector3(0.0f, -0.0f, 0.0f);
                    vtxs[1].TextureCoordinate = new Vector2(.0f, 0.0f);
                    vtxs[2].Position = new Vector3(0.0f, 0.0f, 0.0f);
                    vtxs[2].TextureCoordinate = new Vector2(1.0f, 0.0f);
                    vtxs[3].Position = new Vector3(-0.0f, 0.0f, 0.0f);
                    vtxs[3].TextureCoordinate = new Vector2(0.0f, 0.0f);
                    indices[0] = 1;
                    indices[1] = 3;
                    indices[2] = 1;
                    indices[3] = 3;
                    indices[4] = 2;
                    indices[5] = 1;

                }



            }
            HandleInput();
            base.Update(gameTime);
        }

        Vector3 modelPosition = Vector3.Zero;
        // Posición de la cámara para la matriz de vista
        Vector3 cameraPosition = new Vector3(0.0f, 0.0f, 5000.0f);

    //Fucnion para calcualra la altura del terreno
        public float GetAlturaTerreno(float x, float z, List<Vector3> vertices)
        {
            float distanciaMinima = float.MaxValue;
            float altura = 0;

            foreach (Vector3 vertice in vertices)
            {
                float distanciaTmp = Vector2.Distance(new Vector2(x, z), new Vector2(vertice.X,
vertice.Z)); ;

                if (distanciaTmp < distanciaMinima)
                {
                    distanciaMinima = distanciaTmp;
                    altura = vertice.Y;
                }
            }

            return altura;
        }


    #endregion

        #region Dibujar en el juego
```

```
 ///  <summary> /// este metodo se dibujan las texturas, sprites,
 ///   tienen 3 metodos principales, como son el begin, draw y end, entre otros
///  <summary>
 protected override void Draw(GameTime gameTime)
 {
    GraphicsDevice device = graphics.GraphicsDevice;
    GraphicsDevice.Clear(Color.SkyBlue );
    DibuixarTerreny(camera.view, camera.projection);//dibujamos el terreno

    // Establecer renderstates, ya sea para alambre o un dibujo sólido de la canica
    RenderState renderState = GraphicsDevice.RenderState;
    if (isWireframe)
    {
       renderState.FillMode = FillMode.WireFrame;
       renderState.CullMode = CullMode.None;
    }
    else
    {
       renderState.FillMode = FillMode.Solid;
       renderState.CullMode = CullMode.CullCounterClockwiseFace;
    }

    // Crear matrices de cámara, para el giro de objeto.
    float time = (float)gameTime.TotalGameTime.TotalSeconds;

    float yaw = time * 10.4f;
    float pitch = time * 10.4f;
    float roll = time * 10.4f;
    //Vector de posicion de la camara para la canica y me permite aceracar o alejar la canica
    Vector3 cameraPosition = new Vector3(1, 0, 10.0f);

    float aspect = GraphicsDevice.Viewport.AspectRatio;

    Matrix world = Matrix.CreateFromYawPitchRoll(yaw, pitch, roll);
    Matrix view = Matrix.CreateLookAt(cameraPosition, Vector3.Zero, Vector3.Up);
    Matrix projection = Matrix.CreatePerspectiveFieldOfView(1, aspect, 1, 20);

    // Dibuje la corriente primitiva
    GeometricPrimitive currentPrimitive = primitives[currentPrimitiveIndex];
    Color color = colors[currentColorIndex];

    currentPrimitive.Draw(world, view, projection, color);

    // Restablecer el modo de relleno renderstate
    GraphicsDevice.RenderState.FillMode = FillMode.Solid;
    //dibujo el Cuadro de inicio de juego
    spriteBatch.Begin();

     //dibujar el Texto "Volumen" en pantalla en las coordenadas (0,0)
     spriteBatch.DrawString(texto,Volumen , new Vector2(0, 0), Color.Yellow);


    spriteBatch.Draw(Ayuda, AyudaPos, new Rectangle(0, 0, 1200, 800), Color.Azure);
    spriteBatch.Draw(Info, InfoPos , new Rectangle(-100, -100, 1200, 800), Color.Azure);
    spriteBatch.Draw(Titulo, TituloPos,new Rectangle (-350,-300,1200,800), Color.Azure);
```

```
            spriteBatch.DrawString(texto, seleccionado, new Vector2 (850,10), Color.Red );


            spriteBatch.Draw(Barra , new Rectangle( 0,725, 1200, 150), Color.White);
            spriteBatch.Draw(BotonPlay, new Rectangle(195, 760, 50, 50), Color.White);
            spriteBatch.Draw(BotonSalir, new Rectangle(1095, 760, 50,50), Color.White);
            spriteBatch.Draw(BotonAyuda, new Rectangle(500, 760, 50, 50), Color.White);
            spriteBatch.Draw(BotonInfo, new Rectangle(800, 760, 50, 50), Color.White);
            spriteBatch.End();


            //dibujo el menu
            mimenu.Draw(spriteBatch);


            //habillitamos y dibujamos la sombra
            if (enable_bg)
            {
               spriteBatch.Begin(SpriteBlendMode.None, SpriteSortMode.Immediate,
SaveStateMode.SaveState);
               //spriteBatch.Draw(texture_bg, Vector2.Zero, Color.White);
               spriteBatch.End();
            }

            //Render Sprite
            spriteBatch.Begin(SpriteBlendMode.AlphaBlend, SpriteSortMode.Deferred,
SaveStateMode.SaveState);

            spriteBatch.End();

            //Render triangles
            Matrix world1 = Matrix.CreateTranslation(Vector3.Backward );
            Matrix view1 = Matrix.CreateLookAt(new Vector3(0.0f, 10.0f, 8.2f), Vector3.Zero,
Vector3.Up);
            Matrix proj1 = Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(50.0f), 1.333f,
7.1f, 100.0f);

            device.VertexDeclaration = vtx_declaration;
            //Blend mode
            device.RenderState.AlphaBlendEnable = true;
            device.RenderState.SourceBlend = Blend.SourceAlpha;
            device.RenderState.DestinationBlend = Blend.InverseSourceAlpha;

            effect.Texture = texture;
            effect.TextureEnabled = true;
            effect.View = view1;
            effect.Projection = proj1;
            effect.World = world1;
            effect.DiffuseColor = new Vector3(0.21f, 0.21f, 0.21f);//cambia la atenacion de la sombra
            effect.Alpha = alpha;
            effect.Begin();
            foreach (EffectPass pass in effect.CurrentTechnique.Passes)
            {
               pass.Begin();
```

```
            device.DrawUserIndexedPrimitives<VertexPositionTexture>(PrimitiveType.TriangleList,
vtxs, 0, 4, indices, 0, 2);
            pass.End();
        }
        effect.End();

        base.Draw(gameTime);

    }

    //Funcion que hace referencia al tamaño que ocupa el rectangulo para el menu
    Boolean clic_Item(Rectangle rect, int x, int y)
    {
        return (x >= rect.X &&
            x <= rect.X + rect.Width &&
            y >= rect.Y &&
            y <= rect.Y + rect.Height);
    }




    void HandleInput()
    {
        lastKeyboardState = currentKeyboardState;
        lastGamePadState = currentGamePadState;

        currentKeyboardState = Keyboard.GetState();
        currentGamePadState = GamePad.GetState(PlayerIndex.One);


        // Cambio primitiva?
        if (IsPressed(Keys.A, Buttons.A))
        {
            currentPrimitiveIndex = (currentPrimitiveIndex + 1) % primitives.Count;
        }

        // Cambiar el color?
        if (IsPressed(Keys.B, Buttons.B))
        {
            currentColorIndex = (currentColorIndex + 1) % colors.Count;
        }

        // Activar alambre?
        if (IsPressed(Keys.Y, Buttons.Y))
        {
            isWireframe = !isWireframe;
        }
    }
    bool IsPressed(Keys key, Buttons button)
    {
        return (currentKeyboardState.IsKeyDown(key) &&
                lastKeyboardState.IsKeyUp(key)) ||
               (currentGamePadState.IsButtonDown(button) &&
```

```
                lastGamePadState.IsButtonUp(button));
        }


        /// Fucion para Dibujar el terreno

        void DibuixarTerreny(Matrix view, Matrix projection)
        {
            foreach (ModelMesh mesh in terreny.Meshes)
            {
                foreach (BasicEffect effect in mesh.Effects)
                {
                    effect.View = view;
                    effect.Projection = projection;

                    effect.EnableDefaultLighting();

                    // Modificadores de iluminacion del terreno
                    effect.SpecularColor = new Vector3(0.1f, 0.8f, 0.1f);
                    effect.SpecularPower = 15;


                    //Se establece iluminacion  del terreno en las texturas
                    effect.FogEnabled = false ;
                    effect.FogColor = new Vector3(0.5f);
                    effect.FogStart = 3000;
                    effect.FogEnd = 5000;
                }

                mesh.Draw();
            }
        }

        #endregion


    }
}
```

Clase: cámara

Proporciona la vista de mundo virtual, la Manera en que se va mover dentro del el mundo y cuenta con componentes de la cámara, reutilizable párrafo Aplicaciones.

Funciones públicas miembro

	Cámara (el juego del juego, Vector3 pos, objetivo Vector3, Vector3 arriba)
override void	**Inicializar** () *Inicilizamos la posicion de la camara.*
override void	**Actualización** (tiempo de juego GameTime)

Atributos públicos

Vector3	**cameraPosition**
Vector3	**cameraDirection**
Vector3	**cameraUp**
flotar	**velocidad** = 15

Propiedades

Matriz	**ver** [get, set]
Matriz	**proyección** [get, set]

Descripción detallada

La Clase **cámara** proporciona la vista de mundo virtual, la manera en que se va desplazar el Jugador dentro del mundo virtual.Cuenta con componentes de la cámara, reutilizable párrafo Aplicaciones 3D.

La documentación para esta clase fue generada a partir del siguiente archivo:

- Camera.cs

```
#region Bibliotecas
//Librerias de XNA que se utilizaran posteriormente
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
#endregion
namespace laberinto
{
```

```
   /// <summary> /// La clase Camera proporciona la vista de mundo virtual, la manera en que se va a mover
el jugador
   /// dentro del el y c cuenta con omponenetes de la camara, reutilizable para aplicaciones 3D
   /// </summary>

   public class Camera : Microsoft.Xna.Framework.GameComponent
   {
      // Vectores de la camara
      public Vector3 cameraPosition;
      public Vector3 cameraDirection;
      public Vector3 cameraUp;

      // Matriz de la camara
      public Matrix view { get; protected set; }
      public Matrix projection { get; protected set; }

      // Velocidad de moviemiento de la camara
      public float speed = 15;

      // Angulo Maximo de los movientos de la camara
      float totalPitch = MathHelper.PiOver4 / 2;
      float currentPitch = 0;

      // Estado previo del raton al momento de la comprobacion
      MouseState prevMouseState;

      // Constructor, inicializa la vista de la camara
      public Camera(Game game, Vector3 pos, Vector3 target, Vector3 up)
         : base(game)
      {
         cameraPosition = pos;
         cameraDirection = target - pos;
         cameraDirection.Normalize();
         cameraUp = up;
         CreateLookAt();

         projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
                        2,
                        1, 300000);
      }

#region inicializacion de la camara
      /// Inicilizamos la posicion de la camara

      public override void Initialize()
      {
         Mouse.SetPosition(Game.Window.ClientBounds.Width / 2, Game.Window.ClientBounds.Height / 2);
         prevMouseState = Mouse.GetState();

         base.Initialize();
      }
#endregion

      #region Actualizacion Logica de la camara
```

```
        public override void Update(GameTime gameTime)
        {
            // Rotaciones de la camara
            // ----------------------------------------------
            // Rotacion en X
            if (Mouse.GetState().X >= 0 && Mouse.GetState().Y >= 0 &&
                Mouse.GetState().X <= Game.Window.ClientBounds.Width && Mouse.GetState().Y <=
Game.Window.ClientBounds.Height)
            {

                cameraDirection = Vector3.Transform(cameraDirection,
                    Matrix.CreateFromAxisAngle(cameraUp, (-MathHelper.PiOver4 / 150) *
                (Mouse.GetState().X - prevMouseState.X)));

                // Rotacion en Y (limitacion del angulo )
                float pitchAngle = (MathHelper.PiOver4 / 150) *
                  (Mouse.GetState().Y - prevMouseState.Y);

                if (Math.Abs(currentPitch + pitchAngle) < totalPitch)
                {
                    cameraDirection = Vector3.Transform(cameraDirection,
                        Matrix.CreateFromAxisAngle(
                            Vector3.Cross(cameraUp, cameraDirection),
                            pitchAngle));

                    currentPitch += pitchAngle;
                }
            }

            // Volvemos a poner el mouse en el centro
            Mouse.SetPosition(base.Game.Window.ClientBounds.Width / 2,
base.Game.Window.ClientBounds.Height / 2);

            prevMouseState = Mouse.GetState();
            //mando llamar la funcion  CreateLookat para tener la matriz  vista de la camara
            CreateLookAt();

            base.Update(gameTime);
        }




        /// dejar que la camara mire en direccion a la matriz que he calculado
        private void CreateLookAt()
        {
            view = Matrix.CreateLookAt(cameraPosition, cameraPosition + cameraDirection, cameraUp);
        }
#endregion
    }

}
```

Clase: SpherePrimitive

Funciones públicas miembro

SpherePrimitive (GraphicsDevice graphicsDevice)
SpherePrimitive (GraphicsDevice graphicsDevice, flotador de diámetro, la teselación int)

Descripción detallada

La claseSpherePrimitive Geométricas primitivas de la Clase para elaboración de canica primitiva

---

La documentación para esta clase fue generada a partir del siguiente archivo:

- EspherePrimitive.cs
-

```
#region BIBLIOTECAS
//Librerias de XNA que se utilizaran posteriormente
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
#endregion

namespace Primitives3D
{
   ///  <summary> /// La claseSpherePrimitive
   ///  Geométricas primitivas de clase para la elaboración de esferas
   ///  </summary>


   public class SpherePrimitive : GeometricPrimitive
   {

      //Construye una nueva esfera primitiva, con la configuración predeterminada.

      public SpherePrimitive(GraphicsDevice graphicsDevice)
         : this(graphicsDevice, 1, 50)
      {
      }



      //Construye una nueva esfera primitiva,con el tamaño y el nivel de teselación
```

```
public SpherePrimitive(GraphicsDevice graphicsDevice,
                       float diameter, int tessellation)
{
    if (tessellation < 3)
        throw new ArgumentOutOfRangeException("tessellation");

    int verticalSegments = tessellation;
    int horizontalSegments = tessellation * 2;

    float radius = diameter / 2;

    // Comeinzo con un solo vértice en la parte inferior de la esfera
    AddVertex(Vector3.Down * radius, Vector3.Down );

    // Crear anillos de vértices en latitudes cada vez más elevados
    for (int i = 0; i < verticalSegments - 1; i++)
    {
        float latitude = ((i + 1) * MathHelper.Pi /
                          verticalSegments) - MathHelper.PiOver2;

        float dy = (float)Math.Sin(latitude);
        float dxz = (float)Math.Cos(latitude);

        // Crear un solo anillo de vértices en esta latitud.

        for (int j = 0; j < horizontalSegments; j++)
        {
            float longitude = j * MathHelper.TwoPi / horizontalSegments;

            float dx = (float)Math.Cos(longitude) * dxz;
            float dz = (float)Math.Sin(longitude) * dxz;

            Vector3 normal = new Vector3(dx, dy, dz);

            AddVertex(normal * radius, normal);
        }
    }

    //Termine con un solo vértice en la parte superior de la esfera.

    AddVertex(Vector3.Up * radius, Vector3.Up);

    // Crear un ventilador de conexión en el vértice inferior del anillo de latitud inferior.

    for (int i = 0; i < horizontalSegments; i++)
    {
        AddIndex(0);
        AddIndex(1 + (i + 1) % horizontalSegments);
        AddIndex(1 + i);
    }

    // Llenar el cuerpo de esfera con triángulos que unen cada par de anillos de latitud
    for (int i = 0; i < verticalSegments - 2; i++)
    {
        for (int j = 0; j < horizontalSegments; j++)
```

```
            {
                int nextI = i + 1;
                int nextJ = (j + 1) % horizontalSegments;

                AddIndex(1 + i * horizontalSegments + j);
                AddIndex(1 + i * horizontalSegments + nextJ);
                AddIndex(1 + nextI * horizontalSegments + j);

                AddIndex(1 + i * horizontalSegments + nextJ);
                AddIndex(1 + nextI * horizontalSegments + nextJ);
                AddIndex(1 + nextI * horizontalSegments + j);
            }
        }

        // Crear un ventilador que conecta el vértice superior del anillo de latitud superior.
        for (int i = 0; i < horizontalSegments; i++)
        {
            AddIndex(CurrentVertex - 1);
            AddIndex(CurrentVertex - 2 - (i + 1) % horizontalSegments);
            AddIndex(CurrentVertex - 2 - i);
        }

        InitializePrimitive(graphicsDevice);
    }
  }
}
```

Clase: GeometricPrimitive

Funciones públicas miembro

void	**Disponer** () *Libera los recursos utilizados por este objeto.*
void	**Dibujar** (efecto Efecto)
void	**Dibujar** (Matrix mundo, Matrix vista, la proyección de Matrix, color Color)

Funciones de miembro protegido

void	**AddVertex** (Vector3 posición, Vector3 normal)
void	**AddIndex** (int index)
void	**InitializePrimitive** (GraphicsDevice graphicsDevice) Inicializa primitivas
void l void	**Disponer** (bool disposing)

Propiedades

int	**CurrentVertex** [obtener]

Descripción detallada

La Clase **GeometricPrimitive** dibuja las primitivas de tipo, párrafo geométrico la elaboración de la canica, esta Clase Añade Efectos al Modelo.

Documentación de las funciones miembro

void Primitives3D::GeometricPrimitive::Dispose ( ) [inline]

Libera los recursos utilizados por este objeto.

La documentación para esta clase fue generada a partir del siguiente archivo:

- GeometricPrimitive.cs

```
#region BIBLIOTECAS
//Librerias de XNA que se utilizaran posteriormente
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
#endregion

namespace Primitives3D
{
   ///  <summary> /// La clase GeometricPrimitive dibuja las primitivas de tipo
   ///  geometrico para la elaboracion de la canica, esta clase añade efectos al modelo.
  ///  </summary>
   public abstract class GeometricPrimitive : IDisposable
   {
      #region Fields

      // proceso de construccion de modelo primitivo, vertices
      // y datos alamacenados sobre el  CPU manejados en esta lista.
      List<VertexPositionNormal> vertices = new List<VertexPositionNormal>();
      List<ushort> indices = new List<ushort>();


      VertexDeclaration vertexDeclaration;
      VertexBuffer vertexBuffer;
      IndexBuffer indexBuffer;
      BasicEffect basicEffect;
```

```
#endregion

#region Initialization
protected void AddVertex(Vector3 position, Vector3 normal)
{
   vertices.Add(new VertexPositionNormal(position, normal));
}


protected void AddIndex(int index)
{
   if (index > ushort.MaxValue)
      throw new ArgumentOutOfRangeException("index");

   indices.Add((ushort)index);
}


protected int CurrentVertex
{
   get { return vertices.Count; }
}


protected void InitializePrimitive(GraphicsDevice graphicsDevice)
{

   //declaracion para la creacion de vertices,el formato de nuestros datos vertices
   vertexDeclaration = new VertexDeclaration(graphicsDevice,
                              VertexPositionNormal.VertexElements);

   // Crear un  vertex buffer, y copira nuestro vertex en este.
   vertexBuffer = new VertexBuffer(graphicsDevice,
                              typeof(VertexPositionNormal),
                              vertices.Count, BufferUsage.None);

   vertexBuffer.SetData(vertices.ToArray());

   // crear un index buffer, y copira nuestro index en este .
   indexBuffer = new IndexBuffer(graphicsDevice, typeof(ushort),
                           indices.Count, BufferUsage.None);

   indexBuffer.SetData(indices.ToArray());

   // crear efectos basicos, para ser usados en rrenderizacion de primitivas.
   basicEffect = new BasicEffect(graphicsDevice, null);

   basicEffect.EnableDefaultLighting();
   basicEffect.PreferPerPixelLighting = true;
}


// Finalizer.
    ~GeometricPrimitive()
{
```

```
    Dispose(false);
}


/// <summary>
/// Frees resources used by this object.
/// </summary>
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

protected virtual void Dispose(bool disposing)
{
    if (disposing)
    {
        if (vertexDeclaration != null)
            vertexDeclaration.Dispose();

        if (vertexBuffer != null)
            vertexBuffer.Dispose();

        if (indexBuffer != null)
            indexBuffer.Dispose();

        if (basicEffect != null)
            basicEffect.Dispose();
    }
}


#endregion

#region Draw



public void Draw(Effect effect)
{
    GraphicsDevice graphicsDevice = effect.GraphicsDevice;

    // Establecer la declaración de vértice, búfer de vértice, y búfer de índice.
    graphicsDevice.VertexDeclaration = vertexDeclaration;

    graphicsDevice.Vertices[0].SetSource(vertexBuffer, 0,
                                VertexPositionNormal.SizeInBytes);

    graphicsDevice.Indices = indexBuffer;

    //Dibujar el modelo, utilizando el efecto especificado.
    effect.Begin();

    foreach (EffectPass effectPass in effect.CurrentTechnique.Passes)
    {
```

```
            effectPass.Begin();

            int primitiveCount = indices.Count / 3;

            graphicsDevice.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0,
                                                 vertices.Count, 0, primitiveCount);

            effectPass.End();
        }

        effect.End();
    }



    public void Draw(Matrix world, Matrix view, Matrix projection, Color color)
    {
        // set de parametros basicos.
        basicEffect.World = world;
        basicEffect.View = view;
        basicEffect.Projection = projection;
        basicEffect.DiffuseColor = color.ToVector3();
        basicEffect.Alpha = color.A / 100.0f;

        // Set iimppitante  renderstates.
        RenderState renderState = basicEffect.GraphicsDevice.RenderState;

        renderState.AlphaTestEnable = false;
        renderState.DepthBufferEnable = true;
        renderState.DepthBufferFunction = CompareFunction.LessEqual;

        if (color.A < 255)
        {
            //Establecer renderstates para alpha blended rendering.
            renderState.AlphaBlendEnable = true;
            renderState.AlphaBlendOperation = BlendFunction.Add;
            renderState.SourceBlend = Blend.SourceAlpha;
            renderState.DestinationBlend = Blend.InverseSourceAlpha;
            renderState.SeparateAlphaBlendEnabled = false;
            renderState.DepthBufferWriteEnable = false;
        }
        else
        {
            // Set para la prestación de renderstates opaco.
            renderState.AlphaBlendEnable = false;
            renderState.DepthBufferWriteEnable = true;
        }



        // dibujar el modelo, usando efectos basicos.
        Draw(basicEffect);
    }
```

```
        #endregion
    }
}
```

Clase: GeometriaGeneradaProgramaticament

Aquí está una lista de todos los miembros de la clase con enlaces a las clases que pertenecen:

- abultamentTerreny: **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**
- AddVertex (): **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**
- coordenadasVertices: **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**
- escalaTerreny: **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**
- escalaTextura: **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**
- Proceso (): **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**
- terrenyTextura: **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**

Funciones

- AddVertex (): **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**
- Proceso (): **GeometriaGeneradaProgramaticament:: ProcessadorTerrenys**

Funciones públicas miembro

Overrride ModelContent	**Proceso** (Texture2DContent de entrada, el contexto ContentProcessorContext)

Funciones de miembro estático privado

void	**AddVertex** (MeshBuilder constructor, int texCoordId, int w, int x, int y)

Atributos privados

List <Vector3>	**coordenadasVertices** <Vector3> = new List ()
const float	**escalaTerreny** = 50
const float	**abultamentTerreny** = 1164
const float	**escalaTextura** = 1f
const string	**terrenyTextura** = "roques.bmp"

La documentación para esta clase fue generada a partir del siguiente archivo:

- **Laberinto de canicas.cs**

```
#region Bibliotecas
using System.IO;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content.Pipeline;
using Microsoft.Xna.Framework.Content.Pipeline.Graphics;
using Microsoft.Xna.Framework.Content.Pipeline.Processors;

#endregion

namespace GeometriaGeneradaProgramaticament
{
   //Utilice el procesador para generar un terreny 3D a partir de un mapa

   [ContentProcessor]
   public class ProcessadorTerrenys : ContentProcessor<Texture2DContent, ModelContent>
   {
      List<Vector3> coordenadasVertices = new List<Vector3>();
      const float escalaTerreny = 50;
      const float abultamentTerreny = 1164;
      const float escalaTextura = 1f;
      const string terrenyTextura = "roques.bmp";


      //Genera el terreno a partir del mapa
      public override ModelContent Process(Texture2DContent input, ContentProcessorContext context)
      {
         MeshBuilder constructor = MeshBuilder.StartMesh("Laberinto1");

         // Convierte la textura entrante en un float, para facilitar su procesamiento
         input.ConvertBitmapType(typeof(PixelBitmapContent<float>));

         PixelBitmapContent<float> mapaAlçades;
         mapaAlçades = (PixelBitmapContent<float>)input.Mipmaps[0];

         // Crea los vértices del terreno
         for (int y = 0; y < mapaAlçades.Height; y++)
         {
            for (int x = 0; x < mapaAlçades.Width; x++)
            {
               Vector3 position;
```

```
        position.X = (x - mapaAlçades.Width / 2) * escalaTerreny;
        position.Z = (y - mapaAlçades.Height / 2) * escalaTerreny;

        // Establesco la altura según el color de cada píxel
        position.Y = (mapaAlçades.GetPixel(x, y) - 1) * abultamentTerreny;

        // Guardo las coordenadas para su uso posterior
        coordenadasVertices.Add(position);

       constructor.CreatePosition(position);
    }
}

// Crea un material que se asocia  a la textura del terreno
BasicMaterialContent material = new BasicMaterialContent();

string directory = Path.GetDirectoryName(input.Identity.SourceFilename);
string texture = Path.Combine(directory, terrenyTextura);

material.Texture = new ExternalReference<TextureContent>(texture);

constructor.SetMaterial(material);

// Crea un canal de vertices que mantindrà las coordenadas de la textura
int texCoordId = constructor.CreateVertexChannel<Vector2>(
                        VertexChannelNames.TextureCoordinate(0));

// Crea cada  uno  de los triangulos que formen el terreno
for (int y = 0; y < mapaAlçades.Height - 1; y++)
{
   for (int x = 0; x < mapaAlçades.Width - 1; x++)
   {
      AddVertex(constructor, texCoordId, mapaAlçades.Width, x, y);
      AddVertex(constructor, texCoordId, mapaAlçades.Width, x + 1, y);
      AddVertex(constructor, texCoordId, mapaAlçades.Width, x + 1, y + 1);

      AddVertex(constructor, texCoordId, mapaAlçades.Width, x, y);
      AddVertex(constructor, texCoordId, mapaAlçades.Width, x + 1, y + 1);
      AddVertex(constructor, texCoordId, mapaAlçades.Width, x, y + 1);

      AddVertex(constructor, texCoordId, mapaAlçades.Width, x, y);
      AddVertex(constructor, texCoordId, mapaAlçades.Width, x + 1, y + 1);
      AddVertex(constructor, texCoordId, mapaAlçades.Width, x, y + 1);
   }
}

// Paso el ModelProcessor que convertirà el mesh que acabo de generar
MeshContent meshTerreny = constructor.FinishMesh();

ModelContent model = context.Convert<MeshContent, ModelContent>(meshTerreny,
                                    "ModelProcessor");

// Guardo las coordenadas en la propiedad tag del modelo
model.Tag = coordenadasVertices;
```

```
            return model;
        }


        //Ayuda a agrgar un nuevo vertice de un triangulo al MESHBuilder,Asociado a la coordenada de la
textura


        static void AddVertex(MeshBuilder builder, int texCoordId, int w, int x, int y)
        {
            builder.SetVertexChannelData(texCoordId, new Vector2(x, y) * escalaTextura);

            builder.AddTriangleVertex(x + y * w);
        }

    }
}
```

Clase Menu

Esta es la lista completa de miembros para **MenuXNA:: Menu** , incluyendo todos los miembros heredados.

[inline]		
Draw(SpriteBatch spriteBatch) (defined in **MenuXNA::Menu**)	**MenuXNA::Menu**	[inline]
ItemActual (defined in **MenuXNA::Menu**)	**MenuXNA::Menu**	
Menu(Color noSeleccionadoColor, Color seleccionadoColor, SpriteFont sp, Texture2D textura) (defined in **MenuXNA::Menu**)	**MenuXNA::Menu**	[inline]
seleccionarAnterior() (defined in **MenuXNA::Menu**)	**MenuXNA::Menu**	[inline]
seleccionarSiguiente() (defined in **MenuXNA::Menu**)	**MenuXNA::Menu**	[inline]
tamanoItem(int indexItem) (defined in **MenuXNA::Menu**)	**MenuXNA::Menu**	[inline]
totalItems (defined in **MenuXNA::Menu**)	**MenuXNA::Menu**	
Update(GameTime gameTime) (defined	**MenuXNA::Menu**	[inline]

in **MenuXNA::Menu**)

Funciones públicas miembro

Rectángulo	**tamanoItem** (que indexItem)
	Menu (Color noSeleccionadoColor, Color seleccionadoColor, SpriteFont sp, Texture2D textura)
void	**addItemMenu** (String descripcion, Vector2 posicion)
void	**seleccionarSiguiente** ()
void	**seleccionarAnterior** ()
void	**Actualización** (tiempo de juego GameTime)
void	**Dibujar** (SpriteBatch SpriteBatch)

Propiedades

int	**ItemActual** [get, set]
int	**totalItems** [obtener]

Descripción detallada

La clase **Menu** es esencial para diseñar el contenido de los menús del juego aquí se define el número de temas, colores, selecciones del menú y además las posiciones del mismo se generan los textos de selección y se asignan colores de selección.

La documentación para esta clase fue generada a partir del siguiente archivo:

- Menu.cs

```
//Librerias de XNA que se utilizaran posteriormente
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System.Collections;

namespace MenuXNA
{
   /// <summary> /// La clase Menu es escencial para diseñar el contenido de los menus del juego
   /// aqui se defien el umero de temas, colores, selecciones del menu y ademas las posiciones del mismo
   /// se generan los textos de seleccion  y se asignan colores de seleccion
   /// </summary>
```

```
class Menu
{
        private static int MAX = 20; // máximo de Items en el menú
        private List<String> itemsMenu; // lista de los textos de cada Item
        private List<Vector2> posicionItems; // lista de la posición de cada Item
        private List<Double> escala; // es la escala de cada menú
        private Color noSeleccionado;           //Color del Item que no está seleccionado
        private Color seleccionado;                     //Color del Item Seleccionado
        private SpriteFont fuente;      // Sprite para la lera
        private Texture2D check;        // imagen que sirve para poner antes del texto, puede ser
null

        public int ItemActual { get; set; } //es el Index o id del Item actual seleccionado

        // propiedad que muestra el total de items del Menú
        public int totalItems
        {
                get
                {
                        return itemsMenu.Capacity;
                }
        }

        // propiedad que devuelve el tamaño que ocupa el texto
        public Rectangle tamanoItem(int indexItem)
        {
                return new Rectangle((int)posicionItems[indexItem].X,
                        (int)posicionItems[indexItem].Y,
                        (int)fuente.MeasureString(itemsMenu[indexItem]).X,
                        (int)fuente.MeasureString(itemsMenu[indexItem]).Y);
        }




    //funcion para verificar las seleccion del menu

        public Menu(Color noSeleccionadoColor, Color seleccionadoColor, SpriteFont sp,
Texture2D textura)
        {
                if (textura != null)
                {
                        check = textura;
                }
                fuente = sp;
                itemsMenu = new List<string>();
                posicionItems = new List<Vector2>();
                escala = new List<double>();
                noSeleccionado = noSeleccionadoColor;
                seleccionado = seleccionadoColor;
                ItemActual = 0;
        }

        // función para agregar Items al menú
```

```
        public void addItemMenu(String descripcion, Vector2 posicion)
        {
            if (totalItems < MAX)
            {
                itemsMenu.Add(descripcion);
                posicionItems.Add(posicion);
                escala.Add(1.0f);
            }
        }
    //funcion para adelantar las selecciones en el menu

        public void seleccionarSiguiente()
        {
            if (ItemActual < totalItems - 1)
            {
                ItemActual++;
            }
            else
            {
                ItemActual = 0;
            }
        }
    //funcion para regresar las selecciones en el menu

        public void seleccionarAnterior()
        {
            if (ItemActual > 0)
            {
                ItemActual--;
            }
            else
            {
                ItemActual = totalItems - 1;
            }
        }

    //metodo que permite actualizar  el tamaño de la letra y el tiempo de agrandamiento de las
selecciones
        public void Update(GameTime gameTime)
        {
            for (int x = 0; x < totalItems; x++)
            {
                if (x == ItemActual)
                {
                    if (escala[x] < 2.0f)//tamaño de letra agrandada
                    {
                        escala[x] += 0.1 + 20.0f *
gameTime.ElapsedGameTime.Seconds;//tiempo de agrandamiento de letras
                    }
                }
                else if (escala[x] > 1.0f && x != ItemActual)
                {
                    escala[x] -= 0.1 + 20.0f *
gameTime.ElapsedGameTime.Seconds;//tiempo para hacer pequeña la letra
                }
```

```
                }
            }
            //metodo para dibujar en pantalla el menu, con los Sprites de los temas
            public void Draw(SpriteBatch spriteBatch)
            {
                spriteBatch.Begin();
                for (int x = 0; x < totalItems; x++)
                {
                    if (x == ItemActual)
                    {
                        Vector2 p = posicionItems[x];
                        p.X -= (float)(escala[x] / 2);
                        p.Y -= (float)(escala[x] / 2);
                        if (check != null)
                        {
                            spriteBatch.Draw(check, new Vector2(p.X -
check.Width, p.Y), Color.Orange );
                        }
                        spriteBatch.DrawString(fuente, itemsMenu[x], p, seleccionado,
0.0f, new Vector2(-10, 10), (float)escala[x], SpriteEffects.None, 0);
                    }
                    else
                    {
                        Vector2 p = posicionItems[x];
                        p.X -= (float)(escala[x] / 2);
                        p.Y -= (float)(escala[x] / 2);
                        spriteBatch.DrawString(fuente, itemsMenu[x], p,
noSeleccionado, 0.0f, new Vector2(-100, 10), (float)escala[x], SpriteEffects.None, 0);
                    }
                }
                spriteBatch.End();
            }

        }
    }
```

Clase: Program

Descripción detallada:

La clase Program es el punto principal de entrada para la aplicación, usa el Game para ejecutar la aplicación. La clase fue desarrollada a partir del siguiente código fuente:

```
//Librerias de XNA que se utilizaran posteriormente
using System;

namespace laberinto
{
  static class Program
```

```
    {
        /// <summary>
        /// Es el punto principal de entrada para la plicacion
        /// </summary>
        static void Main(string[] args)
        {
            using (Game game = new Game())
            {
                game.Run();
            }
        }
    }
}
```