Informática Gráfica II

Object-oriented Graphics Rendering Engine

Práctica 1.2: Movimiento del héroe

Alberto Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Objetivo

- En este apartado vamos a mover al héroe por el laberinto
- Solo se puede mover por los huecos, los muros son sólidos
- Tened en cuenta que:
  - El movimiento del héroe (*Sinbad*) y los villanos (*ogrehead*) son iguales
  - Solo cambia la forma de indicar el movimiento
    - El héroe: mediante teclado
    - Los villanos: mediante un algoritmo
  - **La forma de aplicarlo, es la misma**
- Sabiendo esto, podéis estructurar el código en las clases correspondientes
  - Usad herencia y polimorfismo

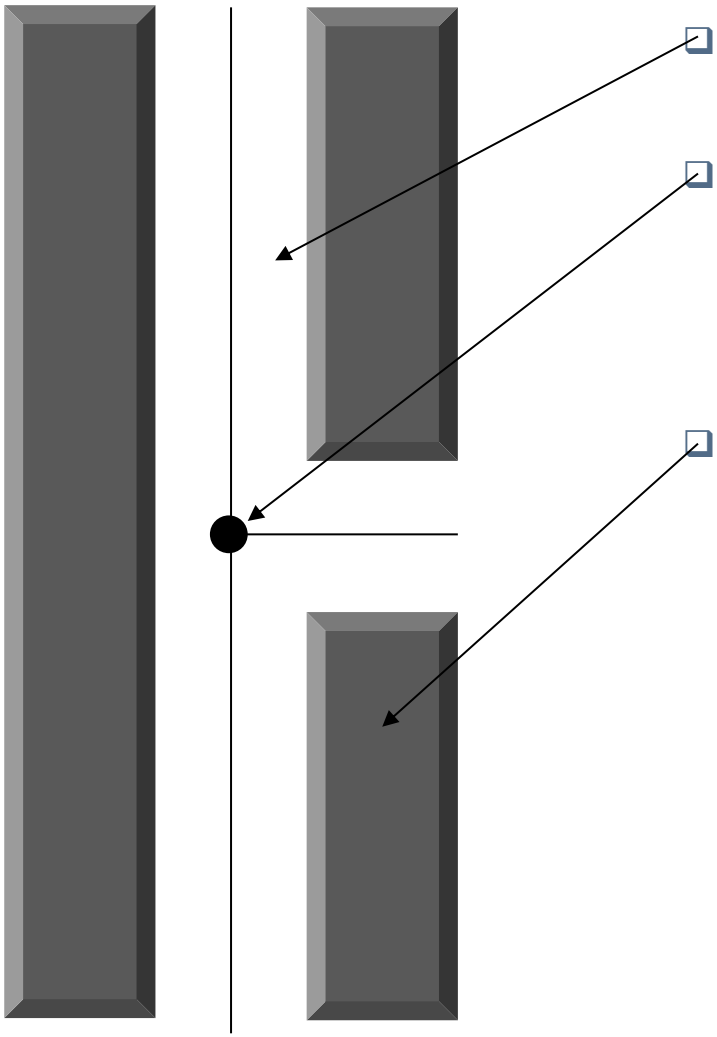
Gestión del Input y actualización del estado

- ¿Cómo sabemos qué tecla se ha presionado?
- La clase encargada de gestionar los eventos hereda de `InputListener`
  - Implementa el método `keypressed`
  - Debe añadirse el objeto correspondiente como *observer* con el método `addInputListener`
    - Este método está en `IG2AppApplicationContext` (Ver Tema 3, slide 49)
- `InputListener` también contiene el método `frameRendered`
  - Se invoca cuando se ha renderizado un frame
  - Útil para actualizar el estado del juego y mover al héroe (si procede)
- Adicionalmente se puede hacer uso de la clase `FrameListener`
  - Contiene los métodos `frameEnded`, `frameRenderingQueued` y `boolframeStarted`
  - Tenemos más control ya que sabemos cuándo se empieza a renderizar el frame

Gestión del Input y actualización del estado

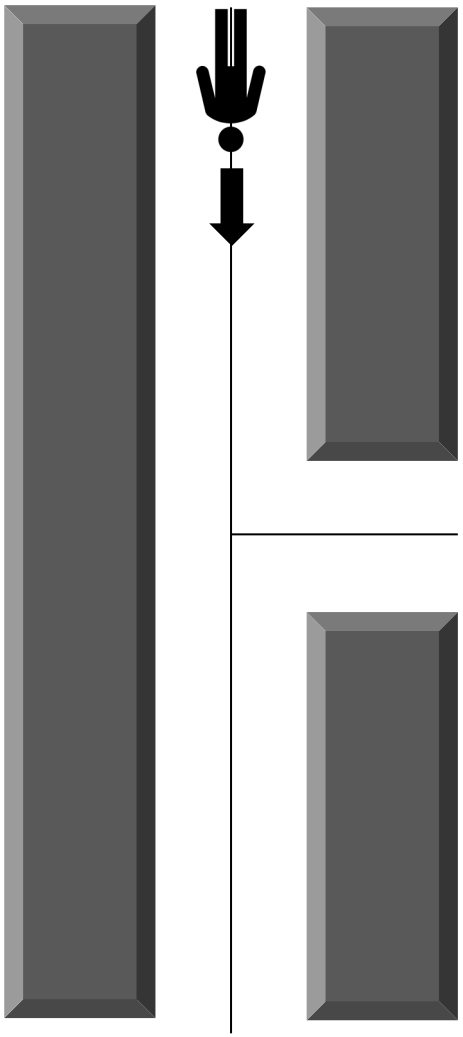
- El movimiento del héroe
  - No puede quedarse parado
    - Salvo que se choque de frente con un muro y bloquee su trayectoria
  - Sólo tendrá en cuenta la **última tecla pulsada**
- ¿Cómo sabemos cuándo podemos cambiar de dirección?
  - Dos formas
    - El héroe siempre se mueve por el centro del pasillo
      - Sabemos cuándo llegamos a una intersección
      - Calculamos si hay que girar
        - Si es el caso, calculamos si es posible girar
        - Si no es posible, se continúa el movimiento en la misma dirección
      - Si no hay que girar, se continua el movimiento en la misma dirección
    - Utilizar AABBs
      - No es necesario mover al héroe por el centro del "pasillo"
      - Más complicada la lógica para los giros
      - Es más difícil que el movimiento sea fluido

Movimiento del héroe (Ejemplo 1)



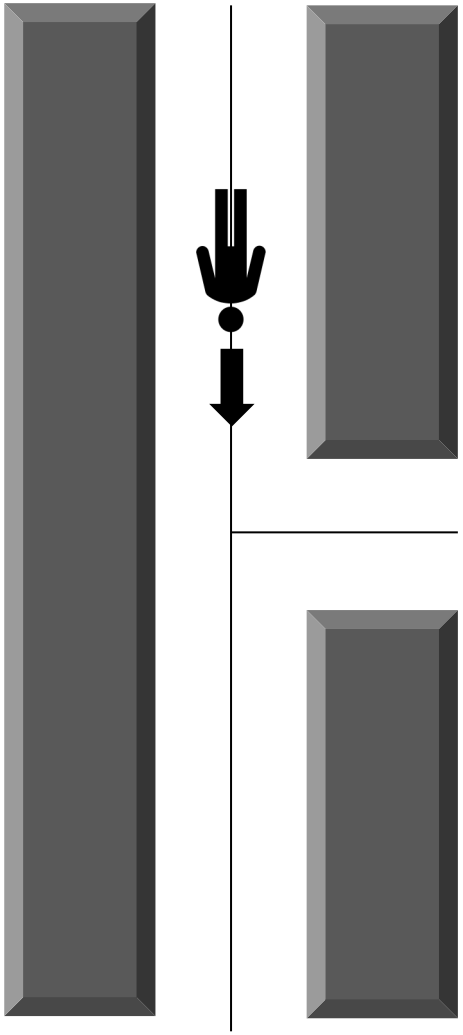
- El héroe se mueve por los huecos
  - La línea (fina) representa la trayectoria
- En los puntos de intersección se hace el cálculo
  - ¿Hay que girar?
    - Sí: Calcular si se puede
    - No: No se hace nada
- El héroe no puede traspasar muros

Movimiento del héroe (Ejemplo 1)



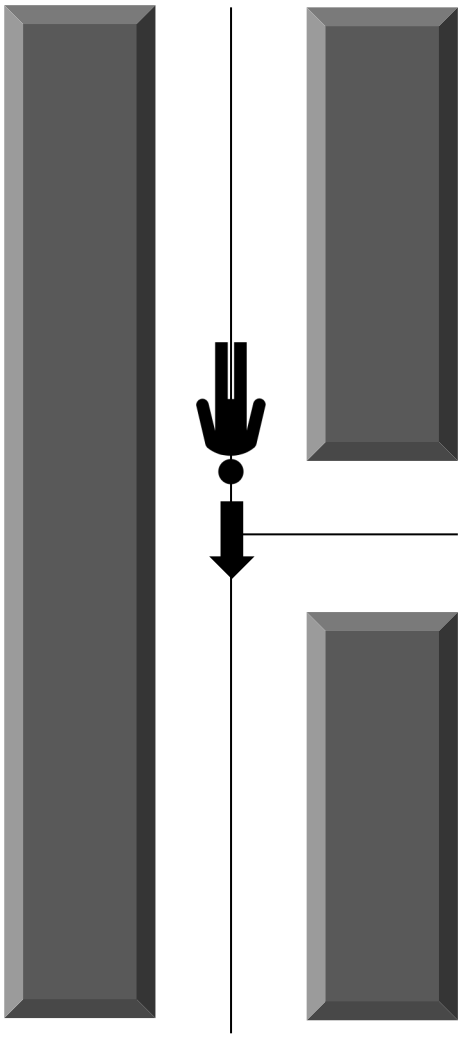
- El héroe se mueve hacia abajo

Movimiento del héroe (Ejemplo 1)



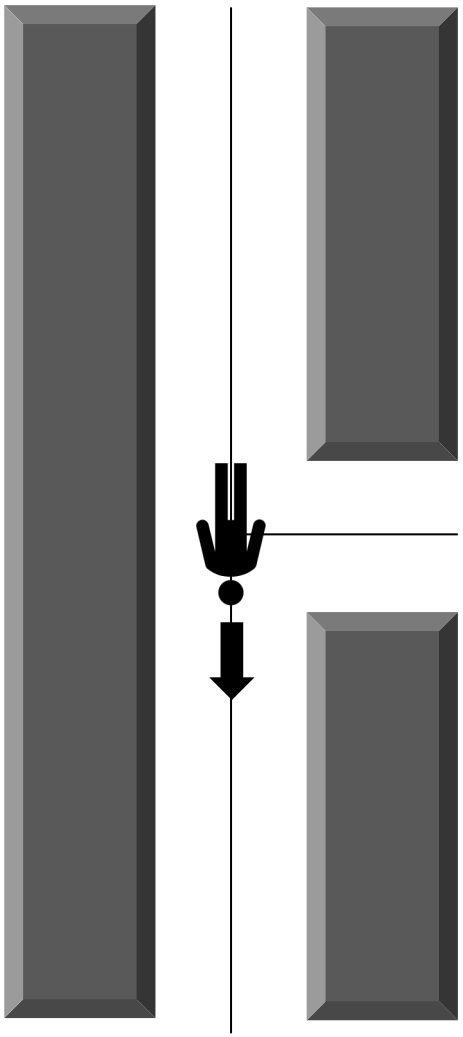
- El héroe se mueve hacia abajo
  - Presionamos la tecla →
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = →

Movimiento del héroe (Ejemplo 1)



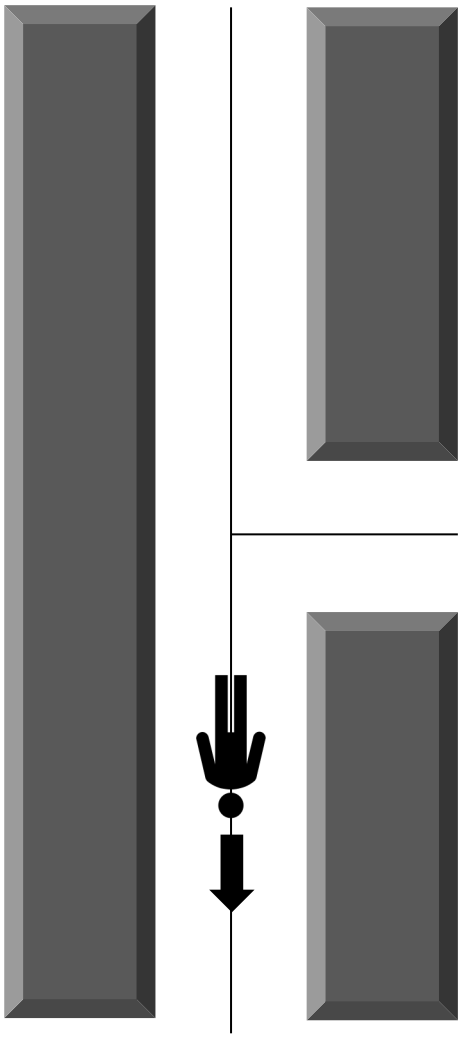
- El héroe se mueve hacia abajo
  - Presionamos la tecla →
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = →
  - Presionamos la tecla ←
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = ←

Movimiento del héroe (Ejemplo 1)



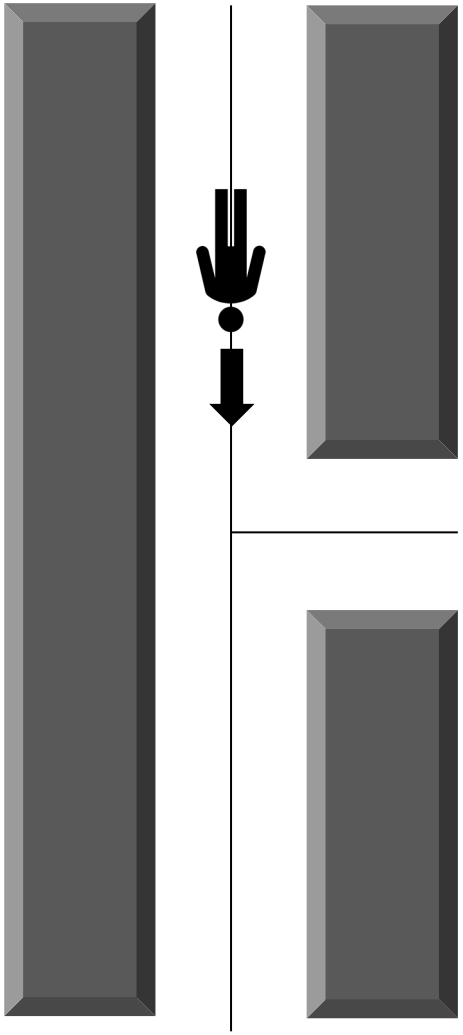
- El héroe se mueve hacia abajo
  - Presionamos la tecla →
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = →
  - Presionamos la tecla ←
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = ←
  - Llegamos a una intersección
    - ¿Podemos movernos a ← ?
    - NO: Seguimos…

Movimiento del héroe (Ejemplo 1)



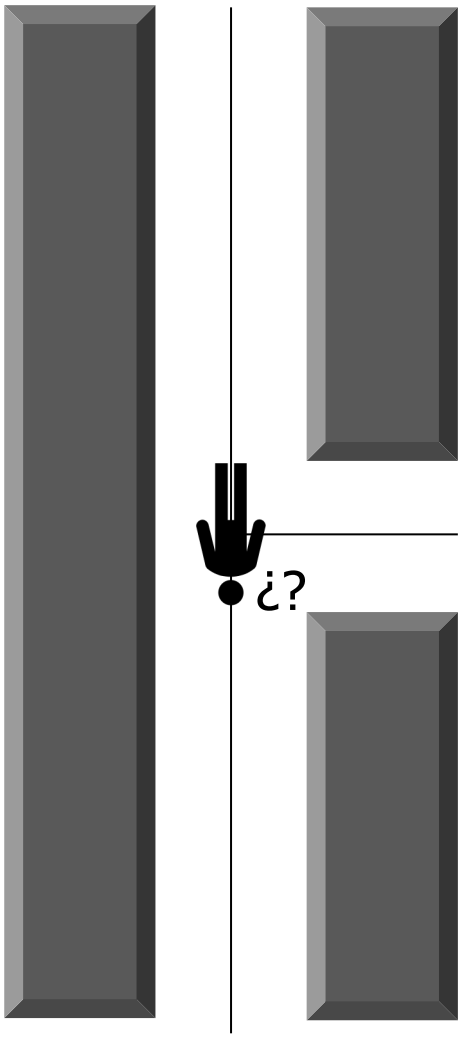
- El héroe se mueve hacia abajo
  - Presionamos la tecla →
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = →
  - Presionamos la tecla ←
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = ←
  - Llegamos a una intersección
    - ¿Podemos movernos a ← ?
    - NO: Seguimos…

Movimiento del héroe (Ejemplo 2)



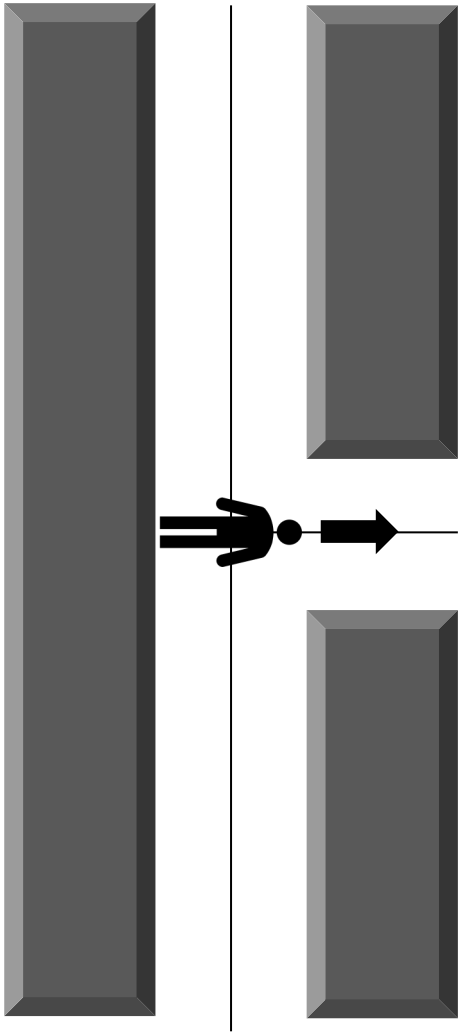
- El héroe se mueve hacia abajo
  - Presionamos la tecla →
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = →

Movimiento del héroe (Ejemplo 2)



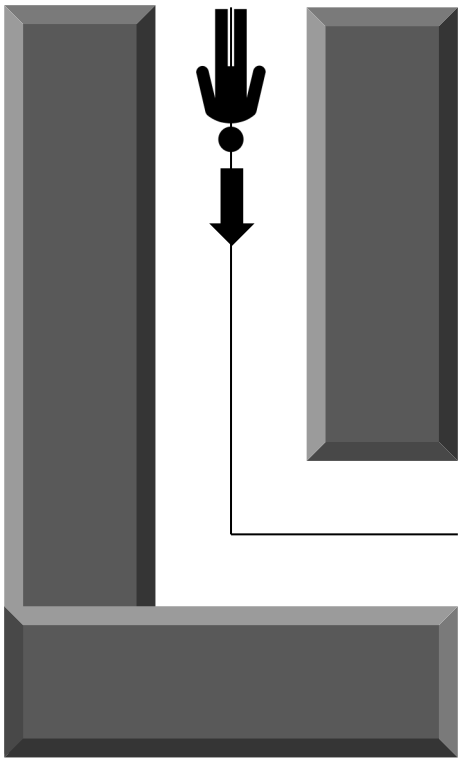
- El héroe se mueve hacia abajo
  - Presionamos la tecla →
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = →
  - ¿Estamos en una intersección?
    - SÍ: Calculamos…
    - ¿Podemos movernos a →?

Movimiento del héroe (Ejemplo 2)



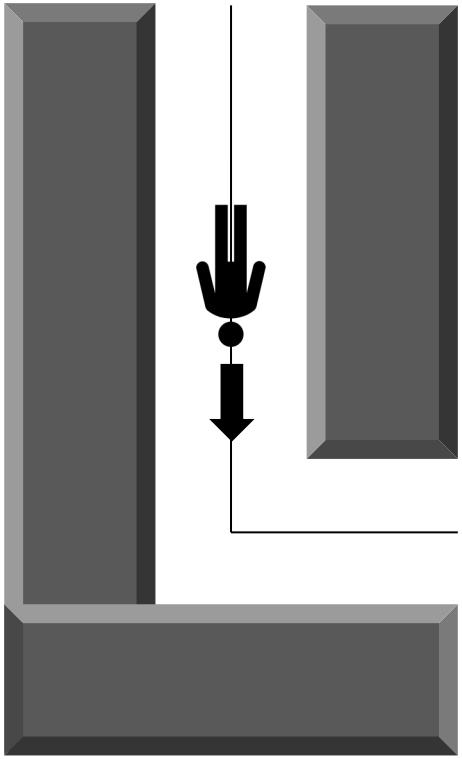
- El héroe se mueve hacia abajo
  - Presionamos la tecla →
  - ¿Estamos en una intersección?
    - NO: No se modifica la trayectoria
    - Actualizamos al héroe
      - Tecla pulsada = →
  - ¿Estamos en una intersección?
    - SÍ: Calculamos…
    - ¿Podemos movernos a →?
      - Sí: Rotamos

Movimiento del héroe (Ejemplo 3)



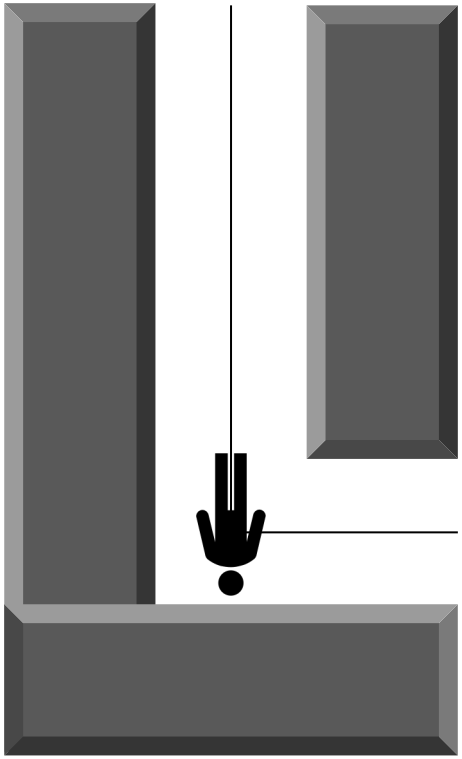
- El héroe se mueve hacia abajo

Movimiento del héroe (Ejemplo 3)



- El héroe se mueve hacia abajo
- Continúa su movimiento
  - No hay tecla pulsada
  - O cualquiera pulsada, menos →

Movimiento del héroe (Ejemplo 3)



- El héroe se mueve hacia abajo
- Continúa su movimiento
  - No hay tecla pulsada
  - O cualquiera pulsada, menos →
- Chocamos contra el muro
  - El héroe se para
- Sólo podemos movernos → o ↑

Implementación del movimiento

- La clase `IG2Object` tiene el método `getOrientation()`
  - Devuelve un `Vector3` con la orientación del personaje (héroe o villano)
- Podemos saber la nueva "posible" dirección con la tecla pulsada
  - Usad las constantes de Vector3:
    - `Vector3::UNIT_X`
    - `Vector3::NEGATIVE_UNIT_X`
    - `Vector3::UNIT_Z`
    - `...`
- La posición del personaje también la conocemos
  - `IG2Object::GetPosition()`
- Podemos pedirle al laberinto si el siguiente bloque en la nueva dirección es *traspasable*
  - Ojo aquí la estructura de clases!
  - Puede ahorrar mucho código y esfuerzo

Implementación del movimiento

- Sólo queda calcular el ángulo de giro
- Se simplifica con el uso de `Quaternions`
- La clase `SceneNode` tiene el método

```
void rotate(const Quaternion& q, TransformSpace relativeTo = TS_LOCAL);
```

- La idea es proporcionar el `Quaternion` que contenga el giro que queremos
- ¿Cómo calculamos el giro?

```
Vector3 newDirVector = this->getNexDirVector();

Quaternion q = this->getOrientation().getRotationTo(newDirVector);
```

  - Siendo `newDirVector` un `Vector3` con la nueva dirección después de girar
  - El `Quaternion` **q** contiene el giro
  - Lo pasamos como parámetro a rotate()

Control de colisiones

- Cuando Sinbad se come una perla…
  - Ésta debe desaparecer
  - Incrementamos la puntuación
- ¿Cómo calculamos que ambas entidades "se tocan"?
  - Con las AABBs
- Tenemos la AABB de Sinbad
  - Y también la de la perla
- En la clase `IG2Object` tenemos el método que nos devuelve su AABB

```
const AxisAlignedBox& IG2Object::getAABB();
```

- La clase `AxisAlignedBox` contiene el método

```
inline bool intersects(const AxisAlignedBox& b2);
```

  - El cual calcula cuándo se cruzan dos AABBs