Sistemas Gráficos

grado en ingeniería informática

Pac-Man 3D

Diseño de la aplicación

Autor
Vladislav Nikolov Vasilev

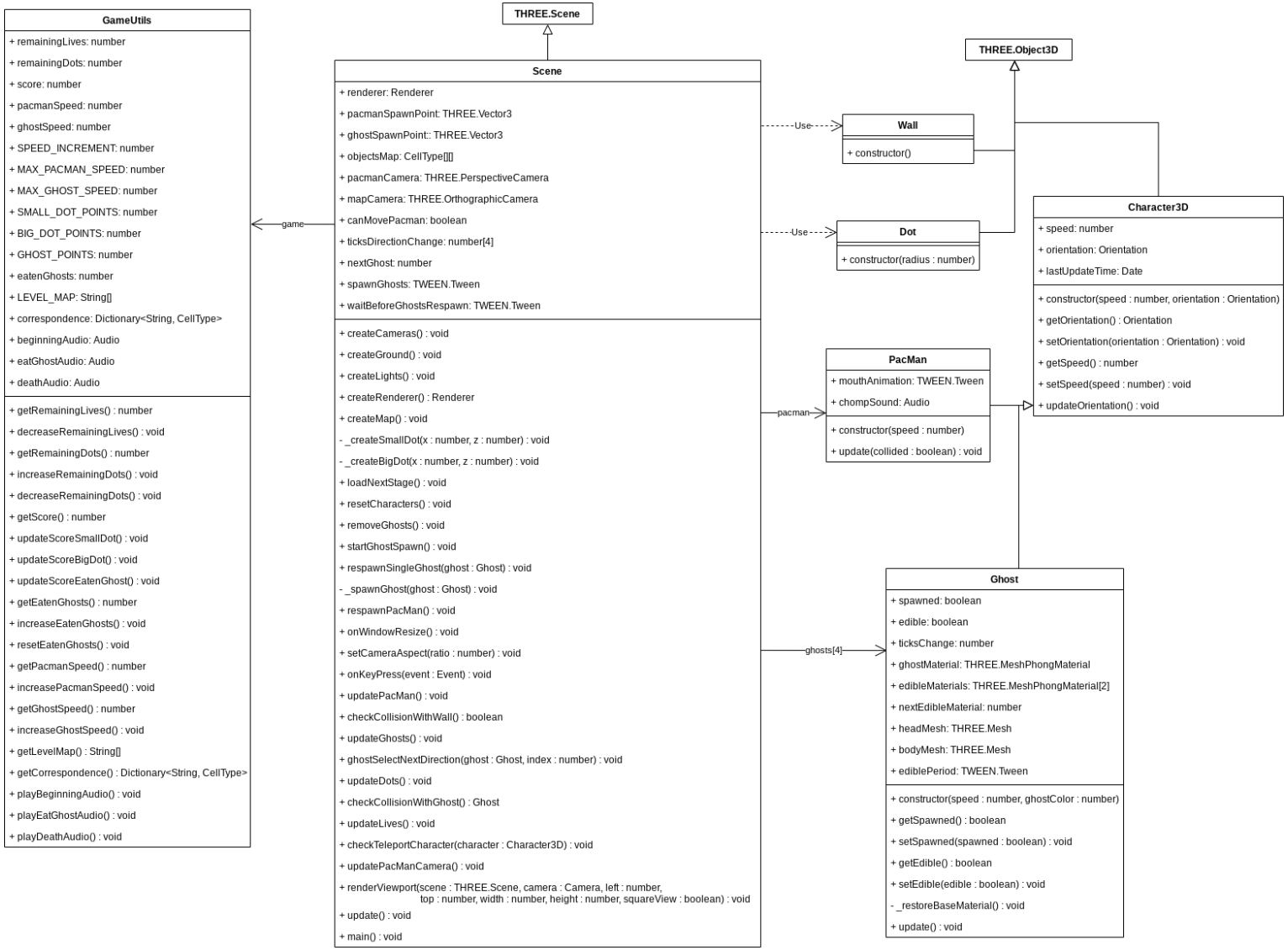
Rama
Ingeniería del Software

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

1. Diagrama de clases



Partiendo del diagrama de clases anterior, vamos a describir las clases para que se entienda qué representa cada una de ellas.

1.1. Clase `Character3D`

Esta clase representa un personaje 3D, heredando de la clase `THREE.Object3D`. El personaje tiene un atributo `speed`, que se refiere a la velocidad a la que se mueve por segundo; un atributo `orientation`, que indica hacia donde está orientado el personaje (arriba, abajo, izquierda o derecha); y un atributo que indica el tiempo en el que se realizó la última actualización del personaje, es decir, en qué momento se llamó por última vez al método `update()` de la clase correspondiente. De esta forma se puede conseguir que los personajes se muevan de forma independiente a la cantidad de fotogramas que sea capaz de renderizar el ordenador, tal y como se ha estudiado en clase.

El método más destacable es `updateOrientation()`. Este método se llama desde el método `update()` de las clases derivadas y se encarga de rotar al personaje de manera que mire en la dirección que le indique el atributo `orientation`.

1.2. Clase `PacMan`

Esta clase representa al personaje `Pac-Man`. Hereda de la clase `Character3D`, con lo cuál hereda sus métodos y atributos. Tiene dos atributos, que son `mouthAnimation` y `chompSound`. El primero es la animación de la boca, la cuál consiste en rotar dos semiesferas en el eje Z cada una en un sentido. Cuando ambas llegan al final del movimiento, comienzan a realizarlo en sentido contrario. Dicho movimiento se repite indefinidamente, deteniéndose solamente cuando el personaje colisiona con un muro. El otro atributo es el sonido del personaje al moverse. Dicho sonido se reproduce siempre y cuando el personaje se esté moviendo, es decir, que no haya colisionado con un muro.

El método `update()` recibe un parámetro, el cuál indica si el personaje ha colisionado contra un muro (verdadero) o no (falso). En caso de que valga verdadero no se actualiza la posición del personaje , se detiene la animación de la boca si no estaba ya detenida y se pausa el sonido del movimiento. En caso contrario, se actualiza la posición en función de la orientación del personaje, se reproduce el sonido y se actualiza la animación de `Tween`, continuando con esta o reanudándola en función de si estaba pausada o no anteriormente.

1.3. Clase `Ghost`

Esta clase representa a un fantasma del juego. Hereda de la clase `Character3D`, con lo cuál hereda sus métodos y atributos. En este caso merece la pena comentar casi todos los atributos propios, ya que tienen una importancia significativa:

- El atributo `spawned` es un booleano que indica si el fantasma ha aparecido en el escenario

o no.

- El atributo `edible` indica si el fantasma es comestible o no.
- `ticksChange` es un atributo que se utiliza en la animación que se lanza cuando el fantasma es comestible. Básicamente, una vez que haya pasado el 70 % del tiempo de la animación, cada 100 *ticks* o actualizaciones de la animación el material del fantasma va a ir cambiando, dando la sensación de intermitencia. El material va a ir cambiando entre el azul oscuro y el blanco.
- `ghostMaterial` es el material base del fantasma.
- El atributo `edibleMaterials` es un *array* que contiene los dos materiales que tiene el fantasma mientras puede ser comido: el azul oscuro y el blanco.
- El atributo `nextEdibleMaterial` es el índice del siguiente material que se va a utilizar en la animación. Empieza con el valor 0, refiriéndose por tanto al azul oscuro, y después se incrementa para hacer referencia al blanco. Después va cambiando de valor cada 100 *ticks* como se ha explicado anteriormente.
- `bodyMesh` y `headMesh` son los *mesh* que forman el cuerpo y la cabeza del fantasma, respectivamente. Es necesario guardarlos, ya que el material de estos se va a ver modificado cuando el fantasma pasa a ser comestible y cuando deja de serlo.
- Por último, `edibleMaterial` es la animación que controla el momento en el que el fantasma es comestible. Tiene una duración de 8 segundos y se activa cuando el `Pac-Man` se come un punto grande. Puede terminar antes de los 8 segundos si el jugador se come al fantasma. Una vez que se acaba, el fantasma deja de ser comestible.

La clase tiene un par de *setters* y *getters*, además del método `update()`, el cuál se encarga de controlar el movimiento del fantasma en función de la orientación. No obstante, de todos ellos destacan dos. Uno de ellos es `setEdible(edible)` el cuál cambia el valor del atributo `edible` e inicia o detiene la animación `ediblePeriod` en función de dicho valor, además de establecer el material del fantasma al correspondiente y de inicializar otros valores, como por ejemplo `nextEdibleMaterials` y `ticksChange`. El otro es `_restoreBaseMaterial()`, el cuál, como su propio nombre indica, restaura el material base del fantasma. Este es un método auxiliar utilizado por el anterior y por la animación una vez que esta haya sido completada.

1.4. Clase `Wall`

Esta clase hereda de `THREE.Object3D` y representa un muro del juego, el cuál es de color azul oscuro. Su constructor simplemente crea una caja de tamaño $1 \times 1 \times 1$ y la posiciona sobre el suelo.

1.5. Clase `Dot`

Esta clase hereda de `THREE.Object3D` y representa o bien un punto pequeño o uno grande en función del radio que se utilice a la hora de crear un *mesh*. Un punto pequeño tiene un radio de 0.1 unidades, mientras que uno grande tiene un radio de 0.2 unidades.

1.6. Clase `GameUtils`

Esta es una clase que contiene toda la información del juego: puntuación del jugador, puntuación que proporciona cada elemento del juego, número de fantasmas que se ha comido el jugador mientras estos son comestibles, vidas restantes, puntos restantes en el mapa por comer, velocidades de los personajes y sus velocidades máximas, audios que se utilizan en determinados momentos de la partida (inicio de la partida, al comerse a un fantasma y al morir) y el mapa del juego junto con su correspondencia a tipos de celdas.

La mayoría de sus métodos son *getters*, además de tener otros métodos que le permiten modificar algunos de los atributos. Hay unos pocos que son de especial interés, como podría ser el caso de los métodos que incrementan la velocidad de los personajes, ya que comprueban que no se ha superado el máximo antes. El incremento de velocidad es de 0.5 cada vez que se completa un nivel. La velocidad inicial del `Pac-Man` es de 3 unidades por segundo, mientras que la de los fantasmas es de 2.5. La velocidad máxima del `Pac-Man` es de 5.5 unidades por segundo, mientras que la de los fantasmas es de 5. De esta forma, el personaje siempre es algo más rápido que los fantasmas.

Otro método interesante es el de aumentar la puntuación del jugador cuando se ha comido a un fantasma. Recordemos que el primer fantasma proporciona 200 puntos, el segundo 400, el tercero 800 y el cuarto 1600. Para obtener estos valores se utiliza la siguiente fórmula:

$$score_{ghost} = 2^{eatenGhosts-1} \times GHOST\_POINTS \tag{1}$$

donde `GHOST_POINTS` es la puntuación base de un fantasma, la cuál es de 200. El restulado de esa expresión se suma a la puntuación del jugador.

1.7. Clase `Scene`

Esta es la clase que contiene la escena

Referencias

[1] Texto referencia
`https://url.referencia.com`