* Las velocidades (de vehículos y troncos) en el archivo de entrada están expresadas en píxeles por segundo. Sin embargo, interesa convertir internamente la velocidad a píxeles por ciclo de actualización, para simplemente sumar la velocidad a la posición en el update. Es una regla de tres sencilla.
* La constante Game::FRAME\_RATE de la plantilla está explicada con «*se actualiza el juego cada tantos milisegundos*», pero esa es la definición del periodo de actualización, no de la frecuencia de actualización. Siendo precisos, la descripción debería decir «*se actualiza el juego tantas veces cada segundo*» o el atributo se debería llamar FRAME\_PERIOD. En la práctica, no hay mucha diferencia, porque un periodo de 30 milisegundos da una frecuencia de 33 hercios.
* Si usáis una HomedFrog inicialmente oculta como forma de detectar que el jugador ha llegado a una casa (como propuso Adrián en clase), podéis extender el enumerado Collision::Type con un nuevo valor HOME (o similar) para indicar este tipo de colisión y que la rana pueda actuar en consecuencia. Una complicación con esta estrategia es que en la posición de la casa pueden coincidir la HomedFrog oculta y una avispa, pero la colisión con la segunda debería tener precedencia sobre la primera.
* He añadido al enunciado una especie de diagrama de secuencia como explicación adicional del flujo de llamadas en las colisiones. El método Game::checkCollision tendrá que iterar sobre los objetos en sus respectivos vectores para comprobar si hay colisión con alguno de ellos.
* El protocolo (que explique en clase de palabra) para manejar la aleatoriedad de las avispas puede ser el siguiente:
  1. Al empezar el juego se obtiene aleatoriamente el tiempo en el que aparecerá la primera avispa (tal vez relativo a SDL\_GetTicks), que se puede almacenar como un atributo de Game.
  2. Cuando se haya cumplido ese plazo, Game::update se dará cuenta y creará una nueva avispa con un tiempo de vida obtenido con getRandomRange. A la vez que se crea la nueva avispa se generará aleatoriamente el tiempo en el que aparecerá la siguiente. Se pueden ajustar los rangos de los números aleatorios para que no aparezca dos avispas simultáneamente, aunque la implementación no debe apoyarse en esa restricción.
  3. Cuando haya transcurrido el tiempo de vida de la avispa, esta se considerará muerta, se dejará de pintar y su método isAlive devolverá false en consecuencia.

Se admiten alternativas, pero **la creación de la avispa siguiente debe ser independiente de la destrucción de la anterior**. Además, las avispas se han de almacenar en el juego como un vector. Esto es un poco forzado, pero es para que practiquéis la eliminación dinámica de objetos.

* El generador de números pseudoaleatorios de tipo [mt19937\_64](https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine.html) se ha de declarar como un atributo randomGenerator de la clase Game. En el constructor de Game se construirá este objeto con su semilla no determinista obtenida como se explica en el enunciado.
* El retorno cíclico de los vehículos y los troncos conviene que sea ligeramente distinto para que todo encaje. Los vehículos regresan del borde exterior al borde interior opuesto, mientras que los troncos saltan del borde exterior al borde exterior opuesto. En el enunciado estaba mal el dibujo, pero ahora está corregido. Ambos saltos se pueden implementar fácilmente con una suma de constante a la coordenada *x* de la posición.