**Descripción de métodos empleados en el diagrama UML del juego “The Quest Finder”.**

***Para la clase Coleccionable:***

El método spawn(): void se encarga de hacer que al momento de instanciar al objeto, éste se instancie en una zona de manera aleatoria.

El método coleccion(): void que de hecho, es un método abstracto, define el comportamiento que tendrá un objeto de la clase Coleccionable al momento de ser coleccionado por el jugador, de ésta manera, en el mismo objeto, se definen las propiedades que va a modificar gracias a su atributo “valor”.

***Para la clase Flecha, extendida de la clase Coleccionable:***

El método de moverX(): void ayuda a que la flecha se mueva en el eje de las x, esto gracias a que cuando el jugador la dispara, la flecha debe moverse una distancia antes de alcanzar a un objetivo o recorrer su distancia máxima.

El método moverY(): void hace que la flecha, gracias a su gravedad descienda para que de ese modo no solo tenga un movimiento horizontal.

Cabe recalcar que la flecha al ser instanciada al inicio del nivel no tendrá ningún daño (daño en 0s) pero que al momento de que es instanciada por parte del jugador, la flecha ahora sí tiene un daño.

***Para la clase Solido:***

La clase sólido en sí, funciona más que nada para que el jugador se encuentre navegando por el mapa gracias a las colisiones que éste tipo de objetos presentará.

***Para la clase Puerta, extendida de Sólido:***

Cuenta con el método siguienteNivel(): void el cual envía al jugador al nivel siguiente (en caso de que exista) si el jugador colisiona con la misma y cuenta con por lo menos una llave.

***Para la clase Personaje:***

El método moverX(): void ayuda a que el personaje pueda moverse en el eje de las X, sumándole la velocidad en X a la posicicionX del mismo.

El método moverY(): void ayuda a que el personaje se mueva en el eje de las Y en forma de “caída”, ya que hace que el personaje caiga gracias a su variable de gravedad.

El método de colision(): void es un método muy importante, ya que ayuda a estar revisando las colisiones que tiene el personaje con otros objetos.

También cuenta con el método atacar(): void que hace que el personaje ataque generando un daño correspondiente a la variable con el mismo nombre (daño).

El método de invertirImagenX(): void ayuda a poder reutilizar la imagen de los personajes, haciendo que la imagen tenga la dirección hacia la que el personaje se encuentre moviendo.

Finalmente, el método de pararImagen(): void ayuda a que, cuando el personaje deje de moverse, entonces la imagen gif deje de moverse también (que deje de cambiar).

***Para la clase Jugador, extendida de Personaje:***

Se tiene el método defender(): void que hace que el jugador pueda defenderse de los ataques enemigos con su escudo.

El método saltar(): void hace que el jugador pueda saltar.

***Para la clase ArañaGigante, extendida de Personaje:***

El método generarAraña(): void genera la instancia de la clase Araña si no se han instanciado el número máximo de estas por cada instancia de ArañaGigante.

***Para la clase Slime, extendida de Personaje:***

Se tiene el método saltar(): void que hace que el slime se encuentre saltando constantemente de un lugar a otro.

***Para la clase MotorAcciones:***

Se tiene el método lanzarAccion(codigoTelca: int): void que hace que se ejecute una acción de acuerdo al código de la tecla que el método recibe.

Nota: La clase LectorTeclas se encarga de leer el código de las teclas presionadas.

***Para la clase Score:***

La clase Score se encarga de almacenar la puntuación del jugador y, mediante el método reiniciarPuntuacion(): void hace que la puntuación del jugador se reinicie en 0s.