**Etapa de diseño**

A continuación, se enlistarán las distintas clases a usar, así como también los distintos métodos que en un principio se desean implementar en el desarrollo del proyecto final.

**Main:**

1. Incluye la biblioteca "QApplication" de Qt y un archivo de encabezado llamado "Game.h".
2. Declara un puntero llamado "game" que apuntará a un objeto de la clase "Game".
3. Define la función principal "main", que es el punto de entrada de la aplicación.
4. Crea una instancia de "QApplication" llamada "a" para administrar la aplicación gráfica.
5. Crea una nueva instancia de la clase "Game" y la asigna al puntero "game".
6. Muestra la ventana de juego.
7. Inicia el bucle principal de eventos de la aplicación, que gestionará la lógica y la interacción con la interfaz gráfica del juego.

**Clase Game:**

1. Incluye la biblioteca "QApplication" de Qt y un archivo de encabezado llamado "Game.h".
2. Declara un puntero llamado "game" que apuntará a un objeto de la clase "Game".
3. Define la función principal "main", que es el punto de entrada de la aplicación.
4. Crea una instancia de "QApplication" llamada "a" para administrar la aplicación gráfica.
5. Crea una nueva instancia de la clase "Game" y la asigna al puntero "game".
6. Muestra la ventana de juego.
7. Inicia el bucle principal de eventos de la aplicación, que gestionará la lógica y la interacción con la interfaz gráfica del juego.

**Clase Player:**

1. Inclusión de archivos de encabezado y bibliotecas: El código incluye varios archivos de encabezado y bibliotecas necesarios para el funcionamiento del programa. Estos archivos contienen las definiciones de clases y funciones que se utilizan.
2. Definición de la clase "Player": Se define una clase llamada "Player", que hereda de la clase QGraphicsRectItem. Esto sugiere que el jugador se representará como un rectángulo en la escena gráfica.
3. Constructor del jugador ("Player::Player"): En el constructor de la clase "Player", se inicializa un objeto bulletsound de la clase QMediaPlayer. Este objeto se utiliza para reproducir el sonido de los disparos. Se cargara un sonido ambiente, próximo a definirse..
4. Manejo de eventos del teclado ("Player::keyPressEvent"): Este método maneja los eventos de teclado cuando el jugador presiona una tecla. Dependiendo de la tecla presionada, ocurren las siguientes acciones:
   * Si se presiona la tecla izquierda (Qt::Key\_Left), el jugador se mueve hacia la izquierda si no alcanza el límite de la pantalla.
   * Si se presiona la tecla derecha (Qt::Key\_Right), el jugador se mueve hacia la derecha si no alcanza el límite de la pantalla.
   * Si se presiona la tecla espacio (Qt::Key\_Space), se crea una instancia de la clase "Bullet" y se agrega a la escena en la posición actual del jugador. Además, se reproduce un sonido de disparo.
5. Método "spawn": Este método se utiliza para crear instancias de la clase "Enemy" y agregarlas a la escena. Esto con el objetivo de generar enemigos en un futuro, desde la clase Game.

**Clase Bullet:**

1. Constructor de la clase "Bullet": En el constructor de la clase "Bullet", se realiza lo siguiente:
   * Se establece la forma y dimensiones del objeto de la bala mediante setRect(). En este caso, se representa como un rectángulo de 10 unidades de ancho y 50 unidades de alto.
   * Se crea un objeto "timer" de la clase "QTimer", que se utiliza para controlar el movimiento de la bala.
   * Se establece una conexión entre el temporizador ("timer") y el método "move()" de la bala. Esto significa que cada vez que el temporizador emite una señal "timeout", se llama al método "move()" de la bala.
   * El temporizador se inicia para que el movimiento de la bala ocurra cada 50 milisegundos.
2. Método "move()" de la clase "Bullet": Este método controla el movimiento de la bala y maneja las colisiones. Las acciones realizadas en este método son las siguientes:
   * Se verifica si la bala colisiona con algún otro objeto en la escena mediante "collidingItems()". Esta información se almacena en la lista "colliding\_items".
   * Se itera a través de los elementos colisionados y se comprueba si son de tipo "Enemy" utilizando la función "typeid". Si se detecta una colisión con un enemigo:
     + Se incrementa el puntaje del juego llamando a la función "increase()" en la instancia global "game->score".
     + Se eliminan tanto el objeto de la bala como el enemigo colisionado de la escena.
     + Se liberan los recursos de memoria al eliminar los objetos.
   * La bala se mueve hacia arriba en el eje Y mediante "setPos(x(), y() - 10)".
   * Si la bala sale de la escena (su posición Y + su altura es menor que 0), se elimina la bala y se liberan los recursos.

**Clase Enemy (y Newenemy):**

1. Constructor de la clase "Enemy" y “Newenemy#: En el constructor de la clase se realizan las siguientes acciones:
   * Se genera un número aleatorio entre 0 y 699 (700 excluido) utilizando la función "rand()" para determinar la posición inicial en el eje X del enemigo. Esto asegura que el enemigo aparezca en una ubicación aleatoria en la parte superior de la pantalla.
   * Se establece la forma y dimensiones del objeto del enemigo mediante "setRect()". En este caso, se representa como un rectángulo de 50 unidades de ancho y 50 unidades de alto.
   * Se crea un objeto "timer" de la clase "QTimer", que se utiliza para controlar el movimiento del enemigo.
   * Se establece una conexión entre el temporizador ("timer") y el método "move()" del enemigo. Cada vez que el temporizador emite una señal "timeout", se llama al método "move()" del enemigo.
   * El temporizador se inicia para que el enemigo se mueva cada 50 milisegundos.
2. Método "move()" de la clase Este método controla el movimiento del enemigo. Las acciones realizadas en este método son las siguientes:
   * Se establecen parámetros para el movimiento (sea MRUA o MRU) .
   * Se calculan las nuevas coordenadas del enemigo utilizando funciones trigonométricas (seno y coseno) para crear el movimiento circular.
   * Se establece la nueva posición del objeto enemigo.
   * Se incrementa la velocidad angular para ajustar el movimiento circular del enemigo.
   * Si el enemigo sale de la parte inferior de la pantalla (posición Y mayor que 600), se disminuye la vida del jugador, se elimina el enemigo de la escena y se liberan los recursos.

Clase Score:

1. **Constructor de la clase "Score"**: El constructor de la clase "Score" se encarga de realizar las siguientes acciones:
   * Inicializa la puntuación (score) a cero.
   * Configura el texto que se mostrará en la interfaz de usuario. Utiliza el método "setPlainText" para establecer el texto en "Score: 0", lo que muestra la puntuación inicial.
   * Establece el color del texto en azul utilizando "setDefaultTextColor(Qt::blue)".
   * Configura la fuente del texto utilizando "setFont(QFont("times", 12))", lo que establece la fuente en "times" con un tamaño de 12 puntos.
2. **Método "increase()"**: Este método se utiliza para aumentar la puntuación en el juego. Cuando se llama a este método, se incrementa el valor de la puntuación en 1 y se actualiza el texto mostrado en la interfaz de usuario utilizando "setPlainText". El nuevo valor de la puntuación se muestra en pantalla, lo que refleja el progreso del jugador.
3. **Método "getScore()"**: Este método permite obtener el valor actual de la puntuación. Se utiliza para consultar la puntuación actual en otros lugares del programa, como para determinar si el jugador ha alcanzado ciertos objetivos o para mostrar la puntuación en diferentes partes de la interfaz

Clase Health:

1. **Constructor de la clase "Health"**: El constructor de la clase "Health" realiza las siguientes acciones:
   * Inicializa la salud (health) del jugador en 3. Esto implica que el jugador comienza con 3 vidas.
   * Configura el texto que se mostrará en la interfaz de usuario. Utiliza el método "setPlainText" para establecer el texto en "Health: 3", lo que muestra la cantidad inicial de vidas.
   * Establece el color del texto en rojo utilizando "setDefaultTextColor(Qt::red)".
   * Configura la fuente del texto utilizando "setFont(QFont("times", 12))", lo que establece la fuente en "times" con un tamaño de 12 puntos.
2. **Método "decrease()"**: Este método se utiliza para disminuir la salud del jugador, lo que generalmente ocurre cuando el jugador pierde una vida. Cuando se llama a este método, se decrementa el valor de la salud en 1 y se actualiza el texto mostrado en la interfaz de usuario utilizando "setPlainText". El nuevo valor de la salud se muestra en pantalla, reflejando la cantidad actual de vidas restantes.
3. **Método "getHealth()"**: Este método permite obtener el valor actual de la salud del jugador. Se utiliza para consultar la cantidad de vidas restantes en otros lugares del programa, como para verificar si el jugador aún tiene vidas o para mostrar la salud en diferentes partes de la interfaz.