Puño Bomba Planeta.

Historia:

El general FourGlass, tirano galáctico que vive en un planeta muy cercano, ha decidido eliminar al planeta de la raza sork ya que son estos una raza altamente evolucionada que se presentan como posible obstáculo del general ante su plan de ser el dictador de todo el universo, es por esto que el general ha decidido enviar a la mejor de sus naves de guerra a bombardear y destruir el planeta zork.

El planeta sork, como métodos de protección ante posibles ataques enemigos ha creado un sistema de Puños galácticos que golpean cualquier objeto que posea una trayectoria hacia el planeta, también han creado un agujero negro con la finalidad de absorber cualquier objeto o proyectil que sea lanzado contra su planeta.

Misión del Jugador.

Uno de los retos del juego a parte de tener que proteger el planeta será absorber las bombas aun cuando la trayectoria de estas haya sido modificada ya sea por colisión de las bombas con los puños galácticos o porque estas entraron en un agujero de gusano que las teletransporto de un punto a otro. sí una bomba colisiona más de 2 veces con un guante, esta explotara dejando un efecto de aturdimiento que va a dificultar el movimiento del agujero negro con lo cual el usuario deberá calcular la distancia más apropiada para que aun estando bajo el efecto de aturdimiento pueda evitar la colisión de las bombas con el planeta.

Rondas:

Las rondas de juego estarán determinadas por la cantidad de bombas lanzadas por la nave, cada 10 bombas cuentan como una ronda y la finalidad será sobrevivir la mayor cantidad de rondas posible.

Visualización.

En la ventana de presentación se tendrán 2 botones:

- El primero para iniciar la partida en modalidad fácil en la cual solo se tendrá que intentar evitar la colisión de las bombas con el planeta adicionalmente de que la trayectoria de las bombas podrá ser modificada si estas llegan a colisionar con los guantes galácticos, los cuales tienen la función de destruir los objetos que tengan trayectoria de colisión con el planeta.

- El segundo botón iniciara el juego en la modalidad difícil se tendrán las funciones implementadas de la jugabilidad en fácil más algunas modificaciones como, por ejemplo: Durante la partida se generarán diferentes agujeros de gusano lo cuales pueden alterar la posición de las bombas que entren en estos y enviarlos a un punto diferente dentro del espacio teniendo que el usuario deba preparar sus reflejos al máximo para evitar de cualquier forma la colisión de las bombas con los planetas.

Cuando se pulse uno de los dos botones se cargará el mapa con sus diferentes ítems dependiendo de la modalidad de juego elegida por el usuario.

Si al menos dos bombas logran colisionar con el planeta se terminará al juego y se le mostrará al usuario la cantidad de rondas que logro sobrevivir y se terminará la ejecución del programa.

Físicas:

Dentro de los sistemas físicos que se van a implementar en el juego se tendrán los siguientes.

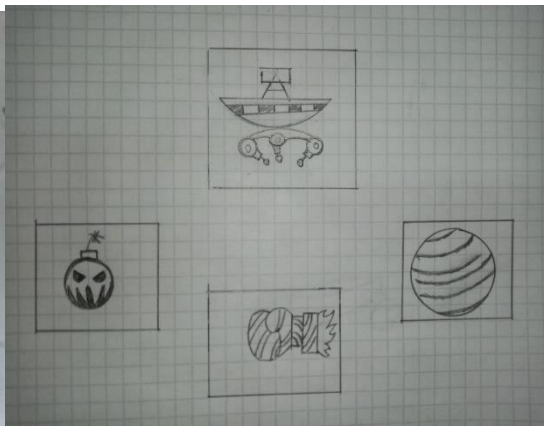
Movimiento parabólico: en cual será implementado cuando uno de los guantes colisione con una bomba modificando así su trayectoria en el eje x más desplazándose siempre en el eje y con dirección hacia el planeta por efecto del campo gravitatorio de este.

Movimiento rectilíneo uniforme y uniformemente acelerado para el movimiento de los guantes y las bombas.

Interacción con el usuario:

Para la parte de la interacción con el usuario se tendrá que el movimiento del agujero negro estará dado por un sensor de ultrasonido el cual al detectar una aproximación mayor a 0 pero menor a 3 cm, desplazará el agujero negro hacia la derecha mientras detecte un objeto en aproximación, si la distancia es mayor a 3 pero menor a 6 se desplazara hacia la derecha mientras detecte el objeto en aproximación.

Bocetos Pantalla del, mapa y los Sprites a implementar:



Lista de Necesidades:

- Clase para crear implementar los métodos y toda la parte gráfica y de interacción de usuario.
- Clase para crear, instanciar e inicializar los diferentes atributos de los objetos a implementar en la escena del juego.
- señales y slots dentro de la clase principal para actualizar el estado de los atributos de cada objeto creado.
- Crear un objeto tipo nave dentro de la clase principal del juego para cargarlo a la escena.
- Crear un vector de objetos tipo proyectil dentro de la clase nave.
- Crear un objeto tipo lanzador dentro de la clase principal del juego para cargarlo a la escena.
- Crear un vector de objetos tipo punch dentro de la clase lanzador.
- Crear un método como slot publico para actualizar el estado de movimiento de los diferentes objetos agregados a la escena del juego.
- Crear un método como slot público para actualizar el estado de los atributos tipo bandera.
- Mapa de pixeles para la implementación grafica de los diferentes objetos en la escena del juego.
- Crear método de Colisiones dentro de la clase principal del juego para verificar el estado de colisión entre los diferentes objetos en la escena del juego.
- Crear método de para cargar uno de los dos niveles del juego de acuerdo con la elección del usuario.

Lista de clases:

- Game.
- Gamebase.
- Nave.
- Portal.
- Lanzador.
- Proyectil.
- Punch.

Lista De Objetos:

- Nave.
- Portal.
- Lanzador.
- Proyectil.
- Punch.

Clases y Objetos.

Clase Game:

La clase Game es la clase principal donde se hará la implementación, instanciamiento e inicialización de los diferentes objetos a implementar en la escena del juego.

La clase Game hereda de forma publica una de las clases de Qt GUI como lo es MainWindow de la cual usa sus diferentes métodos para la creación e implementación de los diferentes Widgets e interfaz gráfica para el juego; Dentro de las diferentes implementaciones que realizaran en la clase Game se tienen las siguientes:

- QKeyEvent.
  Clase con la cual se ejecutará el llamado a los métodos de los diferentes objetos de acuerdo con el evento de teclado registrado.
- QTimer
  Clase con la cual se crearan diferentes variables de tipo Timer que se usaran para los diferentes slots que se tendrán los cuales se tendrá:
  - on_pushButton_clicked : Método a usar para registrar la opción del nivel del de juego seleccionada por el usuario.
  - CargarMundo : Método donde se hará el llamado e instanciamiento e inicialización de los diferentes objetos a implementar en la escena del juego.
  - ReboteDestruk: Método encargado de manejar los estados de movimiento del objeto nave.
  - EjectMove: Este método se encarga de actualizar los estados de los atributos que sirven como bandera para la activación al llamado de los métodos para los objetos agregados a la escena del juego.
  - CargaMov : Este método se encarga de actualizar los estados de movimiento de los diferentes objetos implementados dentro de la escena del juego.
  - CargarPunch : Este método se encarga de cargar los proyectiles de los diferente objetos a la escena del juego.

Clase Gamebase:

Clase que permite la implementación de diferentes métodos y atributos como lo son:

Atributos:

Velx.	Vely.
Dimx.	Dimy.
Posx.	Posy.
Vel.	

Métodos:

getVelx & setVelx	getVely & setVely
getDimx & setDimx	getDimy & setDimy
getPosx & setPosx	getPosy & setPosy
getVel & setVel	

Los métodos y atributos de esta clase se encuentran declarados de manera pública, por lo cual al ser heredad por otra clase, la herencia se debe hacer de manera pública.

Algunas de las clases de Qt que implementa la clase Gamebase son:

- QMainWindow.
- QGraphicsItem.

Clase Portal:

Esta clase hereda de forma publica la clase QObjet, QGrapixItem de las cuales se implementan los métodos tipo GUI con las que se declaran, instancian y actualizan los estados de las diferentes variables propias de la clase Portal.

La clase Portal declara inicializa e instancia los siguientes métodos para el movimiento del objeto Portal:

Move Left y Move Right:

Estos dos métodos se encargan de desplazar un objeto tipo Portal X unidades a la izquierda o derecha de su posición inicial mientras se registre el llamado a uno de estos dos métodos por parte de un objeto tipo Portal creado e instanciado en la clase Game.

Clase lanzador:

Esta clase hereda de forma publica la clase QObjet, QGrapixItem y la clase Gamebase de la cual se implementa sus métodos y atributos. En esta clase también se implementan los métodos de algunas clases Propias de Qt GUI con las cuales se actualizan los estados de las diferentes variables.

La clase lanzador Tiene como finalidad Servir de punto generador de Objetos tipo Punch los cuales se implementarán y serán almacenados en un QVector con el cual luego se agregarán los objetos tipo Punch a la escena Principal del juego.

El objeto tipo lanzador implementa la función disparar con la cual se crea e inicializa un Vector de objetos tipo Punch donde cada objeto recibirá los valores de velocidad y posición para luego ser cargado a la escena principal del juego. La máxima cantidad de elementos de tipo Punch que se tendrán por cada objeto de tipo lanzador será de 3, por lo cual será necesario actualizar constantemente la cantidad de elementos que se tienen por cada lanzador creado e implementado y también rastrear la posición de los objetos tipo Punch para que una vez estos se encuentre fuera de la escena principal del juego sean eliminados tanto del vector como de la escena.

La implementación y actualización de los objetos tipo Punch agregados a la escena se manejará dentro de la clase principal Game a través de Timers y Slosts dispuestos para este fin.

Clase Punch:

Esta clase hereda de forma publica la clase QObjet, QGrapixItem y la clase Gamebase de la cual se implementa sus métodos y atributos. En esta clase también se implementan los métodos de algunas clases Propias de Qt GUI con las cuales se actualizan los estados de las diferentes variables.

En esta clase se instancian e inicializan los valores de las diferentes variables que se encargaran actualizar la velocidad y posición de los objetos tipo Punch agregados a la escena principal del juego, haciendo uso de las ecuaciones de cinemática.

Clase Nave:

Esta clase hereda de forma publica la clase QObjet, QGrapixItem y la clase Gamebase de la cual se implementa sus métodos y atributos. En esta clase también se implementan los métodos de algunas clases Propias de Qt GUI con las cuales se actualizan los estados de las diferentes variables.

Esta clase se encarga de declarar, inicializar e instanciar un objeto tipo Nave a la escena principal del juego, la cual dentro de sus atributos privados tiene un vector de objetos tipo Proyectil con el cual se crean, instancian y agregan Objetos de tipo Proyectil a la escena principal del juego. El objeto tipo Nave ejecuta movimientos repetitivos en el eje x generando Objetos de tipo proyectil en diferentes puntos de la escena grafica del juego.

La implementación y actualización los movimientos del objeto Nave y los objetos tipo Proyectil agregados a la escena principal del juego se hará dentro de la clase principal Game a través de Timers y Slosts dispuestos para este fin.

Clase Proyectil:

Esta clase hereda de forma publica la clase QObjet, QGrapixItem y la clase Gamebase de la cual se implementa sus métodos y atributos. En esta clase también se implementan los métodos de algunas clases Propias de Qt GUI con las cuales se actualizan los estados de las diferentes variables.

En esta clase se instancian e inicializan los valores de las diferentes variables que se encargaran actualizar la velocidad y posición de los Objetos tipo Proyectil agregados a la escena principal del juego, haciendo uso de las ecuaciones de cinemática y movimiento parabólico.

Funciones Comunes de los diferentes Objetos a implementar:

- QTimer:
  Esta función se usa para crear Variables Tipo puntero de la clase Timer con las cuales se actualizarán y ejecutarán algunos métodos que hayan sido definidos e inicializados como slots ya sea de manera pública o privada.

- QRefact:

  Función encargada de determinar y delimitar el tamaño de un objeto grafico dentro de la escena.

- QPixmap:

  Método con el cual se implementará el mapa de pixeles de cada uno de los diferentes objetos para poder agregar a la escena del juego las imágenes de los diferentes espirites y escenarios necesarios para una vista completa de la implementación grafica.

- QPainter:

  Método con el cual se diseñará e implementaran algunos de los métodos y objetos gráficos que se agregaran a la escena del juego.

- Calcular Velocidad:
  Función dentro de la cual se calculará y actualizará el valor de la variable velocidad del objeto que la implemeta.

- Calcular Posición:
  Esta función se encarga de calcular y actualizar la posición del objeto que la implementa.