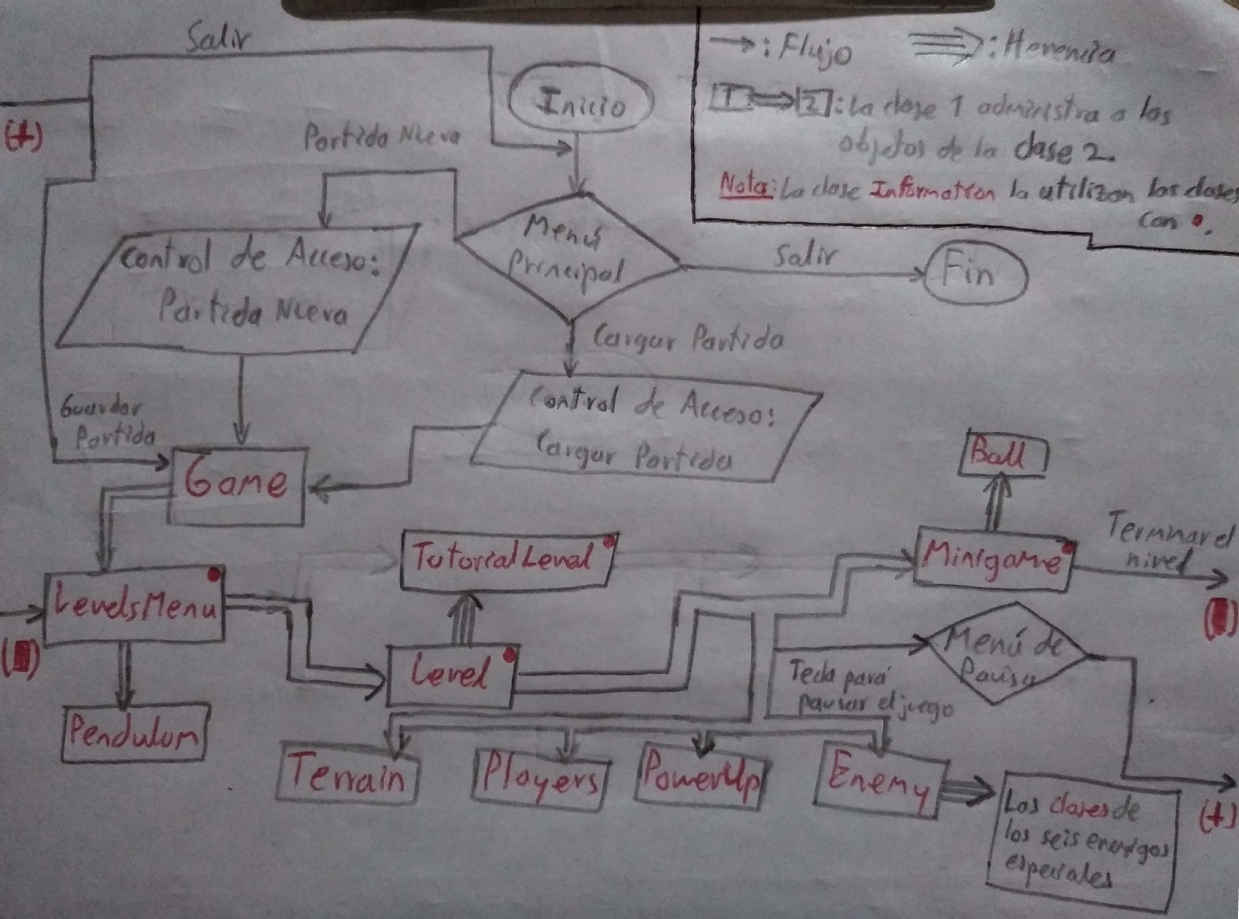
**Clases a Implementar**

Se enunciará y se dará una breve explicación de las clases que se planean implementar, para posteriormente presentar un diagrama ilustrativo de la jerarquía de estas, representando la forma en que unas administrar a otras:

* **Players:** Administrará a ambos jugadores independientemente, de esta forma bastará con mantener el focus sobre este objeto para procesar todas las teclas presionadas por los jugadores. Esta clase heredará las clases **QObject** y **QGraphicsPixmapItem**, pero además tendrá como atributo otro **QGraphicsPixmapItem** correspondiente al segundo jugador.
* **PowerUp:** Modelará las esferas que seguirán distintas trayectorias para otorgar poderes extra a los jugadores, además de todos los algoritmos para la simulación física de estas. Esta clase heredará las clases **QObject** y **QGraphicsPixmapItem**.
* **Terrain:** Administrará la matriz del mapa que almacenará la ubicación de los obstáculos como las trapas o paredes colocadas por los jugadores, además de trabajar en conjunto con el algoritmo para mover a los enemigos. Esta clase utilizará el contenedor **std::array** para la matriz del mapa.
* **Enemy:** Modelará los enemigos; sus patrones de movimiento dependiendo del tipo que sean, su vida restante, la velocidad con la que se mueven, la imagen que llevarán y su animación, las dimensiones, etc. Esta clase heredará las clases **QObject** y **QGraphicsPixmapItem**.
* Los enemigos serán de tres tipos básicos, pero según el nivel aparecerán enemigos especiales, estos últimos heredarán la clase **Enemy**, pero cada uno agregará detalles distintos:
  + **Snail:** Dejará un rastro por dónde camina que aumentará la rapidez de los enemigos que se mueven por encima de este.
  + **Porcupine:** Será como los enemigos normales, pero cuando colisiona contra una pared ambos (el Puercoespín y la pared) se destruyen.
  + **Owl:** Va directo al objeto a defender pasando por encima de los obstáculos y las paredes.
  + **Chamaleon:** Es invisible y no se le puede golpear, pero se hace visible cada cierto periodo de tiempo.
  + **Mole:** Aparecerá muy cerca del objeto a defender.
  + **Vulture:** Al igual que la Lechuza, el Buitre pasa por encima de los obstáculos, pero no se dirige directamente al objeto a defender sino que lo hace en forma de espiral.
* **Information:** Se utilizará para mostrar pequeños letreros en la pantalla, como el número de la oleada al principio de cada una, las recompensas al final del minijuego, el mensaje de victoria o de gameover, etc. Esta clase heredará la clase **QGraphicsPixmapItem**, pero poseerá un atributo de la clase **QGraphicsTextItem**.
* **Level:** Administrará el nivel en general; colocará a los enemigos, colocará a los jugadores, manejará la vida del objeto a defender, llamará los métodos de otros objetos según ciertas condiciones, colocará los poderes extra, llevará el número de la oleada, etc. Esta clase heredará la clase **QGraphicsScene**.
* **TutorialLevel:** Administrará el nivel que servirá de tutorial para los jugadores. Esta clase heredará la clase **Level**.
* **Ball:** Modelará la pelota que se dejará caer en el minijuego, además de todos los algoritmos para la simulación física de esta. Esta clase heredará las clases **QObject** y **QGraphicsPixmapItem**, pero además tendrá como atributo otro **QGraphicsPixmapItem** correspondiente a la garra del minijuego, esto es pues la garra no necesita ninguna funcionalidad en particular más que seguir el mismo MAS que el de la pelota, por lo cual no hace falta modelarla como un objeto independiente.
* **Minigame:** Administrará el minijuego del final de cada nivel. Esta clase heredará la clase **QGraphicsScene**.
* **Pendulum:** Modelará el péndulo que se mencionaba aparecerá en el menú para escoger el nivel, además de todos los algoritmos para la simulación física de este. Esta clase heredará las clases **QObject** y **QGraphicsPixmapItem**.
* **LevelsMenu:** Modelará el menú de la interfaz para escoger el nivel a jugar, además de administrar los métodos del objeto **Pendulum**. Esta clase heredará la clase **QGraphicsScene**.
* **Game:** Almacenará toda la información de la partida en curso, además de administrar el guardado, cargado y control de acceso de las partida.

**Notas:**

1. La clase **Game** estará en sintonía con la interfaz gráfica de Qt para interactuar con el usuario.
2. El menú principal del juego se realizará con la interfaz gráfica de Qt, por lo cual, a excepción del control de acceso, el resto de partes del juego serán objetos heredados de la clase **QGraphicsItem** dentro de otros heredados de la clase **QGraphicsScene**.
3. Se hace uso de la herencia en las clases que heredan otras propias de Qt, y entre clases propias del proyecto en la clase **TutorialLevel** que hereda la clase **Level**, y las seis clases de los enemigos especiales, que heredarán la clase **Enemy**.
4. Hay objetos, como los obstáculos colocados por los jugadores o el rastro dejado por los caracoles enemigos, que aparecerán en pantalla, pero como es suficiente colocarlos mediante un **QGraphicsPixmapItem** y guardar su información dentro de la matriz de la clase **Terrain**, no hace falta definir una clase propia para estos.

Ahora, como se mencionó al principio, se mostrará un diagrama ilustrativo de la jerarquía de las clases según la forma en que unas administran a otras, y de paso un bosquejo del flujo que seguirá la ejecución del programa:

**Cronograma de Actividades**

Las actividades están divididas en nueve bloques que, por medio de una estimación de cuánto tiempo tomaría desarrollar cada actividad, poseen unas fechas límites para su culminación, además de agregar un día extra por posibles retrasos en dos puntos estratégicos del desarrollo del proyecto. En este orden de ideas el cronograma de actividades es el siguiente:

* **Bloque 1**,del 9 al 10 de Julio (Dos días):
  + Crear la clase **Level** y **Terrain** para agregar algunos obstáculos al nivel.
  + Crear la clase **Enemy** y programarla para que se mueva hacia el centro de la pantalla.
  + Programar la interacción de la clase **Enemy** con los obstáculos del nivel.
  + Hacer pruebas hasta que todo funcione correctamente.
* **Bloque 2**,del 11 al 12 de Julio (Dos días):
  + Permitir que los jugadores puedan atacar a los enemigos para bajarles vida y que estos se eliminen de la pantalla cuando mueran.
  + Programar a los enemigos para bajarle vida al objeto a defender cuando llegan hasta él.
  + Programar a los enemigos especiales.
  + Hacer pruebas para verificar que si cumplan con su descripción.
* **Bloque 3**,del 13 al 14 de Julio (Dos días):
  + Crear la clase **PowerUp**, programarla para que las esferas sigan las trayectorias propuestas y que cuando los jugadores las golpeen adquieran un poder extra.
  + Programar los poderes extras.
  + Hacer pruebas para verificar la correcta interacción con los enemigos.
* **Bloque 4**,del 15 al 15 de Julio (Un día):
  + Crear la clase **Information**.
  + Terminar de programar toda la clase **Level** para que administre la aparición de los enemigos según las oleadas, los **PowerUp** y para que se gane o se pierda el nivel. Dejar funcionando el primer nivel.
* **Tiempo Extra**,del 16 al 16 de Julio (Un día):
  + **Opcional:** Si el cronograma se ha cumplido exactamente como se ha previsto, se procederá a sustituir los gráficos provisionales hasta el momento, por los que podrían ser los definitivos
* **Bloque 5**,del 17 al 18 de Julio (Dos días):
  + Crear la clase **Minigame** junto con la clase **Ball** y programar toda la simulación de los dos sistemas físicos del minijuego.
  + Hacer pruebas para verificar su correcto funcionamiento.
  + Al terminar el minijuego proporcionar la información de las recompensas obtenidas mediante la clase **Information**.
* **Bloque 6**,del 19 al 20 de Julio (Dos días):
  + Crear las clases **LevelsMenu** y **Pendulum**; programar toda la simulación del sistema físico.
  + Hacer pruebas para verificar su correcto funcionamiento.
  + Crear la clase **Game** y programarla para guardar la partida al final de un nivel o en medio de este desde el menú de pausa.
  + Hacer pruebas de la interacción con los archivos de texto.
* **Bloque 7**,del 21 al 21 de Julio (Un día):
  + Realizar la interfaz gráfica del control de acceso y vincularlo con la clase **Game**.
  + Realizar la interfaz gráfica del menú principal y agregar la funcionalidad de pasar al control de acceso para comenzar una nueva partida o cargar alguna guardada, además de agregar la posibilidad de guardar durante un nivel.
  + Hacer pruebas para verificar el guardado y cargado de partidas.
* **Tiempo Extra**,del 22 al 22 de Julio (Un día):
  + **Opcional:** Si el cronograma se ha cumplido exactamente como se ha previsto, se procederá a sustituir los gráficos provisionales hasta el momento, por los que podrían ser los definitivos
* **Bloque 8**,del 23 al 24 de Julio (Dos días):
  + Programar de forma similar el resto de niveles basándose en el primero, pero agregando diferentes enemigos en mayor o menor medida, además de los enemigos especiales de cada nivel.
  + Ensamblar todas las partes para tener el flujo del programa definido.
  + Programar el nivel tutorial.
  + Programar el modo de un solo jugador.
* **Bloque 9**,del 25 al 26 de Julio (Dos días):
  + Agregar o mejorar los gráficos del juego que no hayan sido colocados aún o que puedan mejorarse.
  + Depurar todo el proyecto.
  + Documentar el proyecto.