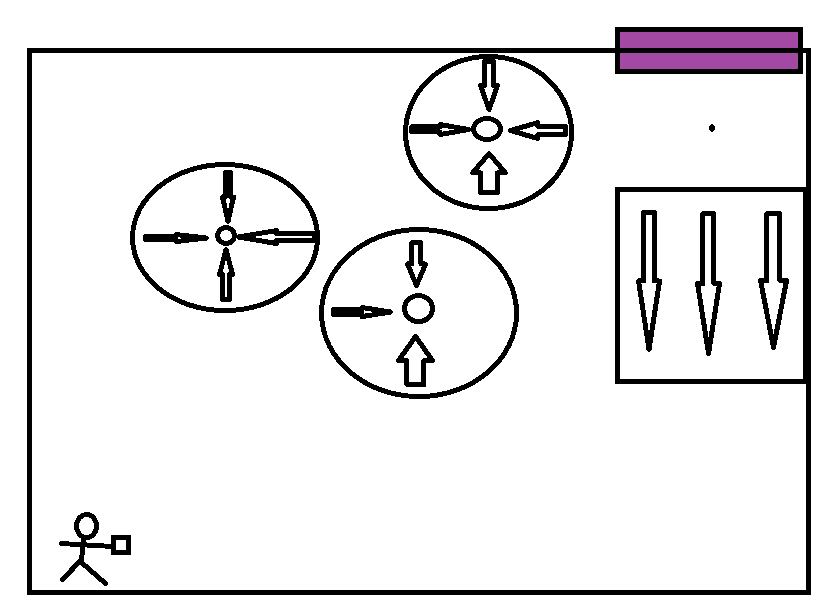
Juego basado en puzles tipo portal, pero en 2D. El jugador tendrá que ingeniárselas para superar las pruebas a propuestas en cada mapa y lograr activar los “checks” de nivel validándolo. El jugador se moverá a través del mapa para hallar la manera de avanzar al siguiente nivel. También se podrá completar el nivel de al menos 2 maneras diferentes.

Será un juego en 2D con enfoque en las físicas y en las interacciones entre los objetos desarrollados.

# **El plan inicial es lograr el siguiente objetivo:**

Tener un personaje movible, que sufra efectos físicos, que pueda controlado por el usuario. El jugador podrá interactuar con diversos elementos a través de un arma de gravitación para intentar lograr completar objetivos por cada uno de los niveles, estos objetivos deberían poder ser logrados de diversas maneras:

Ejemplo:



En este sketch del nivel inicial se pondría una esfera que actúe como objeto a pasar a través del área morada, las áreas con flechas serían áreas con aceleraciones constantes. El jugador deberá usar las interacciones de la esfera con las áreas para desviarla y que lleguen al área.

De esta manera se podría plantear el uso de diferentes objetos (afectados por la gravedad o no) para que se implementen diferentes mapas con diferentes soluciones.

En principio, las interacciones se planean a través de modelados físicos “gravitacionales”, aunque también se plantea el uso de zonas radiales repulsivas.

**Jugador**: Que se pueda mover a través de impulsos de aceleración, tanto de los inputs del que lo controla, como por impulsos de otros objetos.  
**Arma de gravitación**: Que repela y atraiga objetos dinámicos. Inicialmente se podría plantear que, al sostener un clic, se cree una zona de aceleración (invisible) lo suficientemente grande como para atraer al objeto sin necesidad de implementar mucho más que la zona. Que repela se podría lograr con una función que lea el clic y aplique una aceleración lo suficientemente grande para mover al objeto, aunque tal vez sea más adecuado aplicarle una velocidad.

**Zonas de aceleración**: Zonas que apliquen una aceleración constante a los objetos dinámicos que se encuentren en su zona de efecto. Se implemente una opacidad para que el jugador vea la dirección de la zona.

**Objetos Dinámicos:** (Cajas que se puedan mover con el arma, por ejemplo)

**Objetos Estáticos**: A pesar de su nombre, no se plantea que se queden necesariamente en un lugar del mapa siempre, la parte de estático iría más bien ligada al hecho de que no se verían afectados por zonas de aceleración.

**Puntos de Gravedad**: Se podría ver como una zona de aceleración radial, que emule atracción o repulsión. Estos puntos no deberían chocar directamente con los objetos dinámicos, sólo darles aceleraciones.

Segunda iteración:

Se ahondará un poco más sobre las clases usadas y las ideas generales para cada una:

# **ObjetoEstatico:**

Heredará QGraphicsPixmapItem y QObject.

Se usará como clase base para objetos que no serán afectados directamente por la gravedades o aceleraciones de algún tipo. Se espera que tengan colisiones con otros objetos; se podrán escalar para, por ejemplo, hacer paredes, plataformas, etc; tendrá un tamaño y un ángulo de rotación. Guardará también la imagen que se usa en el objeto.

Sus atributos pasarían a ser:

float Angulo

QPointF \*Size: El tamaño del objeto para un posible escalamiento de la imagen original

QPixmap \*Imagen: El Sprite que se usará

QRectF \*Borde: El rectángulo de choque usado para las colisiones

Como métodos :

virtual void CargarSprite(): tal que se pueda usar para las clases hijas para asignarle una imagen a \*Imagen. También se podría plantear el uso de un QList<QPixmap\*> \*Lista, para tener una lista de los sprites usados por el objeto y lograr una animación.

# **Plataforma:**

Hereda ObjetoEstatico.

Su constructor es una copia de ObjetoEstatico, e incluye una variable “colorPlataforma” que se usará para identificar el tipo de plataforma.

Con esta variable se podría pedir en el constructor dicha variable para elegir entre las posibilidades (inicialmente negra y blanca).

Estos objetos chocarían a los objetosMovibles y no los dejarían mover.



En la imagen se pone una plataforma como pared.

# **ObjetoMovible:**

Heredará QGraphicsPixmapITem y QObject.

Estos objetos podrán ser movibles a través de aplicaciones de velocidades y/o aceleraciones.

Su constructor incluye que tipo de objeto será generado (jugadores, cajas, cubos, esferas, etc), una masa, la posición, la velocidad inicial y la aceleración inicial.

Más allá de guardar las variables usadas para calcular los movimientos, también incluirá un QRectF para saber su caja de colisiones, sus sprites, el frame actual, etc. Siendo:

Velocidad, Aceleración, Masa, TipoDeObjeto, Masa, VelocidadMaxima, Friccion, Tamaño, Borde, QList<QPixmap> \*Sprites,

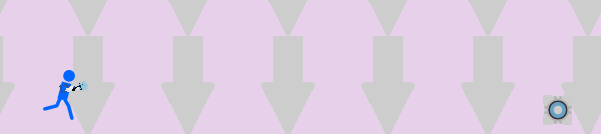
Los posibles métodos:

AumentarVelocidad(), setAceleracion(), setPos() (se crea como método virtual por si los hijos necesitan mover cosas extra en junto a ellos), se crean diferentes getter(), siguienteFrame (para iterar sobre las imágenes y emular un Sprite)



En la imagen se muestran 2 objetos, el de la izquierda, el jugador (Hijo de esta clase) que se moverá por inputs y las diferentes aceleraciones dadas por las zonas gravitacionales. A la derecha está una instancia directa de ObjetoMovible con la variable Cubo para TipoDeObjeto.

Ambos objetos serían afectados por las zonas tal que:



En toda los mapas habría como mínimo la zona gravitacional (la de rosa aunque con opacidad 0 para ser invisible) que emula la gravedad.

# **ZonaGravitacional:**

Heradará de QGraphicsPixmapItem y QObject.

Se espera guardar la información relevante para calcular valores de aceleración aplicados a objetos movibles. Inicialmente se tendría guardado FuerzaGravitacional y la rotación de la zona.

Se calcularía entonces con las funciones trigonométricas las aceleraciones aplicadas en x y y para los objetos dentro de los límites de la zona.

Como atributos tendría:

Int FuerzaGravitacional (para hallar la aceleración aplicada en la zona en función de este valor y la masa del objeto a mover)

float Rotacion (para saber los diferentes componentes de la aceleración)

como métodos tendría getters de los atributos.

Probablemente también se le deba aplicar un setPos() personalizado dado que se planea usar para el Arma gravitacional.

# **Arma:**

Heredará de QGraphicsPixmapItem y QObject

Tendrá 2 tipos de disparos inicialmente, atractivos y repulsivos. En la implementación de los disparos inicialmente se podría usar para el disparo atractivo una ZonaGravitacional que exista mientras se da un clic, dicha zona tendría una fuerza grande e iría desde la posición del arma hasta el primer objetoMovible encontrado. Para el repulsivo, se puede implementar que, mientras el objetoMovible esté pegado al arma, se pueda aplicar una aceleración grande.

Sus atributos podrían ser:

TipoDeDisparo (Atractivo o Repulsivo)

ZonaGravitacional \*ZonaDeEfecto (tal que se cree y actualice constantemente para hallar el primer ObjetoMovible y atraerlo, además, esta zona se crearía sólo mientras una tecla o clic esté presionado, probablemente se deba usar un Signal que haga trigger cuando se de clic y se suelte)

# **Jugador:**

Hereda de ObjetoMovible.

Su constructor es una copia de ObjetoMovible, pero en este se agrega un objeto extra Arma.

También necesita sobreescribir una función virtual heredada de ObjetoMovible (setPos) para que cuando se le aplique también mueva la posición del arma.

Como atributos extra tendría:

Arma \*Pistola

Y métodos adicionales un getPistola() para poder modificarla cuando haga falta

SetPos(): que sobreescriba al de la clase madre para que en esta se mueva también el arma.

En la implementación del jugador también se pondrá que el arma rote en función de la posición del jugador con respecto al mouse.



Jugador y arma.

# **Mapa:**

Atributos:

Tendrá listas de instancias de los objetos descritos usados para cada mapa en su construcción.

Un identificador del mapa actual

Metodos:

Inicialmente se contempla

De reinicio de mapa

CargarMapa()

# **Juego:**

Heredará de QGraphicsView

Sus atributos incluirán:

Timers, las instancias para la pantalla, para el jugador, listas de instancias de mapas

Métodos:

Funciones para la interacción jugador-juego como:

Jugar(): empieza el juego, carga mapas, etc.

Pausar(): Podría implementarse un menú de pausa, todos los timers activos deben ponerse en pausa en este parte.

Despausa(): (Quita el menú de pausa y reanuda los timers)

Funciones para la interacción de diferentes tipos de objetos, ejemplo:

void InteraccionZonas(Zona \*, Objeto \*) que se encargaría de aplicarle una aceleración al objeto, entre otras.

InteraccionParedesPegajosas()

También sobreescribiría funciones virtuales para poder manejar la interacción del jugador con los elementos interactivos del juego. Tales como mousePressEvent(), mouseMoveEvent(), keyPressEvent(), etc.