

CONTENIDOS

CONTENIDOS	1
2.1. Concepto de fuerza	2
2.2. Máquinas simples	2
2.2.1. Palancas	4
2.2.2. Poleas	7
2.2.3. Aparejos	8
2.3. Plano inclinado	10
2.4. Leyes del movimiento (Leyes de Newton)	10
2.5. Diagrama de cuerpo libre	11
2.5.1. Sistema de referencia	12
2.5.2. Cálculo de la aceleración	15
2.5.3. Fuerza de Rozamiento	16
2.5.4. Coeficientes de rozamiento	17
2.6. Tipos de fuerzas	18
2.6.1. Fuerza Elástica	18
2.7. Motor de Física	19
2.7.1. ¿Qué es Box2D?	19
2.7.2. Factories	20
2.8. Entonces, ¿cómo se usa?	20
2.8.1. El mundo de la simulación	20
2.8.2. Objetos de simulación	21
2.9. Armandó el rompecabezas	24
2.9.1. Cómo avanzar en el tiempo	24
2.9.2. La simulación	25
2.9.3. Interfaz de depuración	25
2.10. Box2d y SFML	26
2.10.1. Fuerzas en SFML	29
BIBLIOGRAFÍA	30

2.1. Concepto de fuerza

Para que un cuerpo comience a moverse es necesario aplicarle una *fuerza*. Si no lo hacemos, dicho cuerpo permanecerá en su estado de reposo por tiempo indefinido. Iniciamos un movimiento, por Ejemplo, cuando empujamos un mueble, pateamos una pelota, tiramos de una caja... Es decir, son acciones donde ejercemos una fuerza para que el objeto cambie su estado y comience a moverse. De la misma manera, cuando un cuerpo está en movimiento y el objetivo es detenerlo, tendremos que aplicar una fuerza contraria al movimiento del objeto.

El efecto que produce una fuerza sobre un cuerpo depende del módulo, la dirección y el sentido en que se aplica. Esto significa que la fuerza es una magnitud vectorial y se la representará como a todos los vectores.

Entonces, cuando se aplica una fuerza, puede ocurrir lo siguiente:

- Si el cuerpo está en reposo, comienza a moverse.
- Si el cuerpo está en movimiento, cambia su velocidad (en módulo, dirección o sentido).
- Si el cuerpo está en movimiento, se detiene.

Unidades de fuerza: De acuerdo al Sistema Internacional (SI), la unidad de fuerza es el *Newton* = [N]

Fuerza Peso: Es muy común escuchar que *todo lo que sube, baja*. No importa a qué altura arrojes un objeto porque, en un determinado momento, éste se detendrá y comenzará a descender. La fuerza que hace que los cuerpos descendan es la *fuerza de la gravedad* y se conoce como el *peso* del cuerpo.

El peso de un cuerpo es proporcional a su masa. Esto quiere decir que cuanto mayor sea su masa, mayor será su peso. Es un error frecuente decir que el peso de un cuerpo es 70 kg, cuando en realidad esto hace referencia a su *masa* (magnitud escalar). El peso, al ser una fuerza, tiene unidades de fuerza, por lo que también se

puede utilizar el \vec{Kg} , que se lee *Kilogramo fuerza*.

Por lo tanto, el peso de dicho cuerpo es $70 \vec{Kg}$, que son 686 [N], aproximadamente. La relación matemática que existe entre las dos unidades de fuerza

es la siguiente: $1 \vec{Kg} = 9,8[N]$

¿Cuándo aplicamos una fuerza?

Cuando queremos mover, levantar o bajar un objeto, aplicamos una fuerza. Pero cuando la masa de dicho objeto es muy elevada, por ejemplo, 100 Kg, es difícil levantarlo sin ayuda. Es por ello que, ya desde la Antigüedad, el hombre fue creando sus propias *máquinas simples*, que en la actualidad continúan vigentes, para que lo ayudasen en su trabajo pesado.

2.2. Máquinas simples

Ejemplos de máquinas simples son las palancas, poleas y los engranajes; el plano inclinado; tornos y otros. En esta unidad, nosotros estudiaremos las palancas, las poleas y el plano inclinado.

No obstante, para una mejor comprensión de las máquinas simples, deberemos estudiar previamente el momento de una fuerza y la condición de equilibrio de un cuerpo.

Momento (o torque) de una fuerza

Maquina simple

Es un dispositivo a través del cual se puede modificar el módulo, dirección, sentido y/o punto de aplicación de una fuerza, con el objetivo de obtener algún provecho.

Para ilustrar este concepto, consideremos que tenemos un cuerpo que puede girar en torno a un eje. En Fig. 1 podemos observar un cuerpo rígido sobre el que actúa una fuerza F aplicada sobre el punto a . También es posible ver el punto O sobre el que un Eje atraviesa el cuerpo rígido, y finalmente, está d (perpendicular al eje), que es la distancia entre O y la dirección de F .

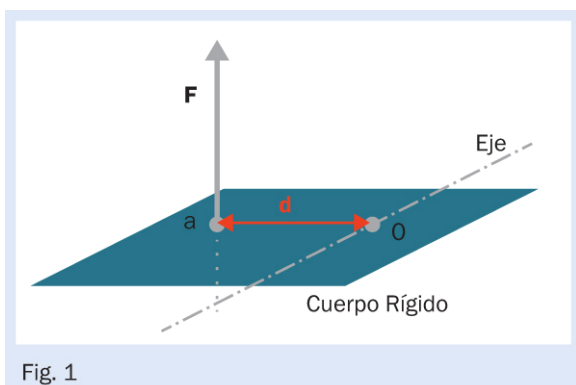


Fig. 1

Así, de acuerdo a esta figura, *momento* es el producto de F por d y se define:

$$M = F \times d = [\text{N} \cdot \text{m}]$$

Unidades de Momento $M = [\text{N}] \cdot [\text{m}]$

Ejemplo 1:

Sobre un cuerpo actúan dos fuerzas, cuyas direcciones se encuentran a diferentes distancias del eje de rotación, siendo los datos:

$F_1 = 20 \text{ [N]}$, $d_1 = 5 \text{ [m]}$ y $F_2 = 15 \text{ [N]}$, $d_2 = 6 \text{ [m]}$.
 El momento de F_1 es: $M_1 = F_1 \times d_1 = 20 \text{ [N]} \times 5 \text{ [m]} = \mathbf{100 \text{ [N.m]}}$.
 El momento de F_2 es: $M_2 = F_2 \times d_2 = 15 \text{ [N]} \times 6 \text{ [m]} = \mathbf{90 \text{ [N.m]}}$.

El momento es una cantidad vectorial, es decir, que puede ser representado por un vector.

Volvamos a la Fig. 1 para representar el vector momento:

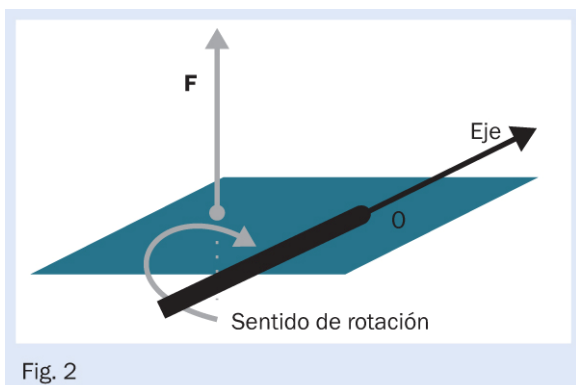
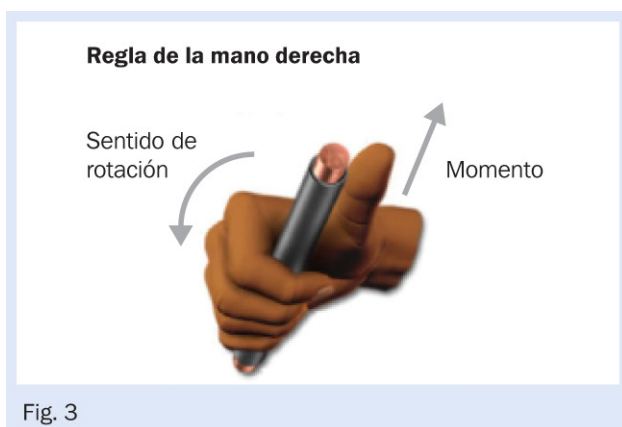


Fig. 2

Como vemos en la Fig. 2, el vector momento tiene como dirección el eje de rotación y el sentido se obtiene por la regla de la mano derecha, como se muestra en la siguiente figura:



Equilibrio de un cuerpo rígido

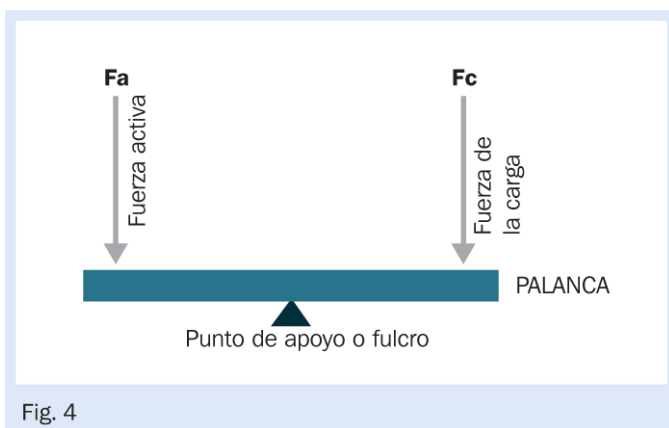
Un cuerpo rígido se encuentra en condiciones generales de equilibrio cuando la sumatoria (Σ) de todas las fuerzas y la sumatoria de todos sus momentos son iguales a cero, es decir:

$$\Sigma \mathbf{F} = 0$$

Σ = símbolo de sumatoria

$$\Sigma \mathbf{M} = 0$$

2.2.1. Palancas



Una palanca es una barra rígida que se encuentra apoyada en un punto sobre el cual puede girar. Hay tres clases de palancas: las de primer género o clase 1; las de segundo género o clase 2, y las de tercer género o clase 3.

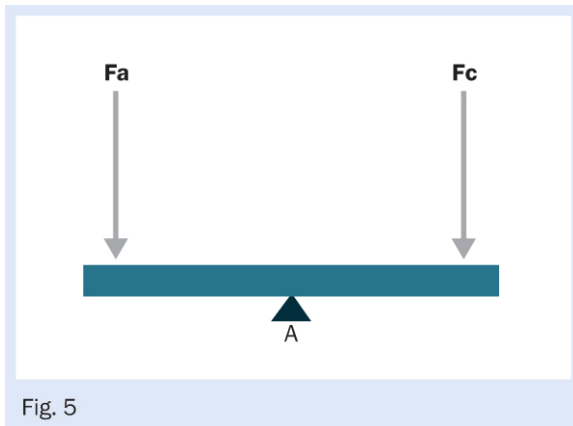
Una palanca consta de:

- *Apoyo o fulcro (A)* Punto de apoyo de la palanca.
- *Fuerza de Carga (Fc) o Resistencia (R)* Fuerza ejercida por la carga.
- *Fuerza Activa (Fa) o Potencia (P)* Fuerza que debe ejercerse para mover la carga.

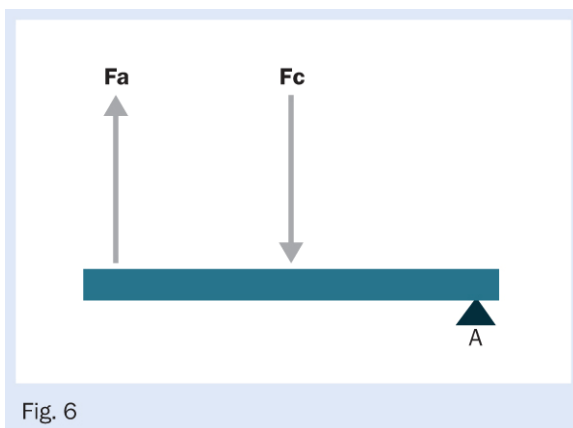
Clases de palancas

Hay tres clases de palancas que se categorizan en función del elemento que se ubica en medio de la misma:

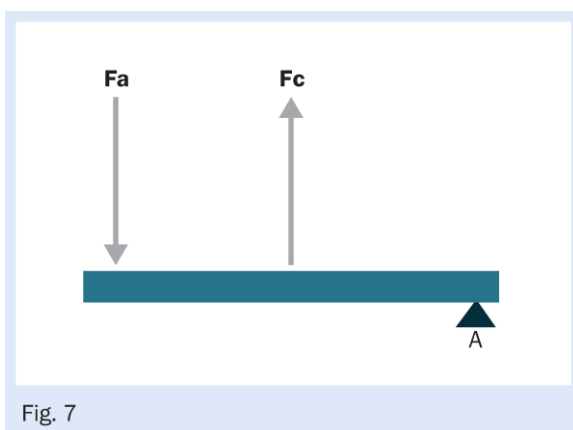
Palancas de primer género o clase 1 El punto de apoyo se encuentra en el medio:



Palancas de segundo género o clase 2 La fuerza de la carga se encuentra en medio de la palanca:



Palancas de tercer género o clase 3 La fuerza activa se encuentra en medio de la palanca:



Condición de equilibrio de la palanca

En cualquiera de las clases de palancas que hemos visto, una de las condiciones de equilibrio es que la sumatoria de los momentos sea igual a cero.

Veamos el siguiente gráfico para una mejor comprensión:

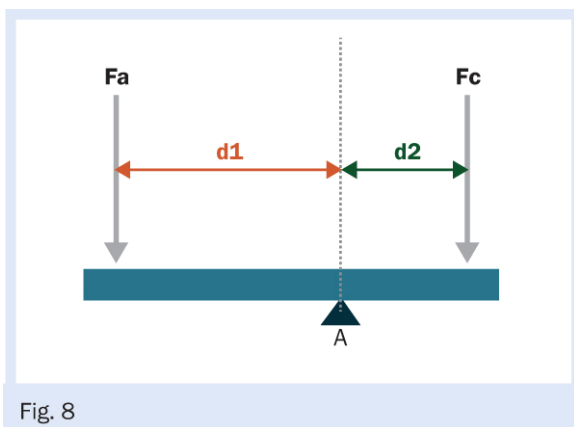


Fig. 8

Como se puede observar, hay dos fuerzas sobre la palanca. Por lo tanto, para que la suma de los momentos de cada fuerza sea cero, el momento de F_1 debe ser igual al momento de F_2 . Así, escribimos que $M_1 = M_2$, y de la ecuación 1 ahora tenemos:

$$F_1 \cdot d_1 = F_2 \cdot d_2$$

Esta ecuación es también conocida como *ley de la palanca*.

Entonces, siempre que se cumpla esta condición, la palanca estará en *equilibrio*.

En caso de que cambie la magnitud de alguna de las fuerzas o su punto de aplicación, la palanca dejará de estar en equilibrio.

Arquímedes, al descubrir las ventajas de la palanca, dijo: “Denme un punto de apoyo y moveré el mundo”.

Ejemplo 2:

Veamos con un ejemplo una aplicación de una palanca de primer género.

Dada la siguiente figura, calcular la fuerza activa (F_a) que se debe ejercer para levantar el bloque de masa m_1 , siendo $P = 1960 \text{ [N]}$, $d_a = 8 \text{ [m]}$ y $d_c = 4 \text{ [m]}$.

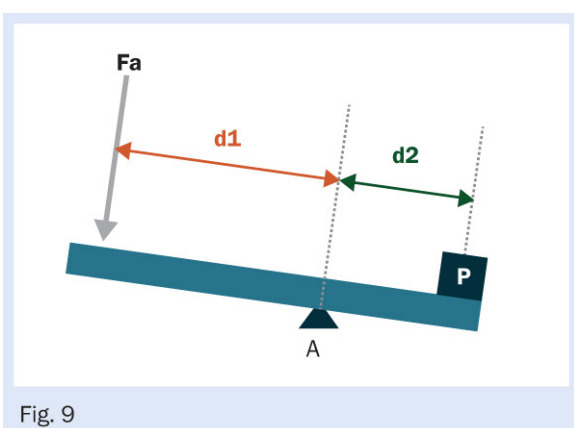


Fig. 9

La fuerza de la carga F_c será el peso del objeto que debemos levantar: $F_c = 1960 \text{ [N]}$

Ahora sabemos que $F_a \cdot d_a = F_c \cdot d_c$

Entonces:

$$F_a = \frac{F_c \cdot d_c}{d_a} = \frac{1960 \cdot 4}{8} = 980 \text{ [N]}$$

Ejemplos de aplicaciones de palancas por género:

Palanca de 1 ^{er} género	Palanca de 2 ^{do} género	Palanca de 3 ^{er} género
		

2.2.2. Poleas

Una polea es un dispositivo mecánico formado por una rueda, montada sobre su eje, y un canal –llamado garganta– en toda su circunferencia para que un cable o sogá produzca el giro de la misma (Fig. 10).

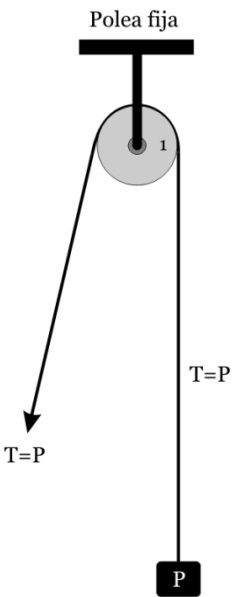


Fig. 10

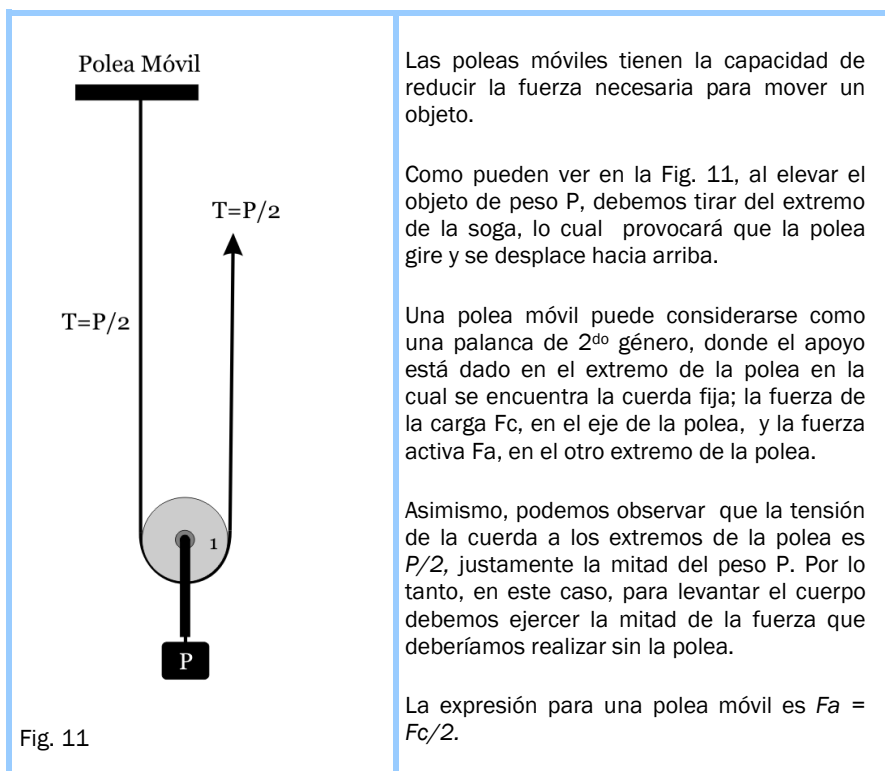
Las poleas fijas no reducen fuerzas, sino que sólo se utilizan para modificar el sentido de una fuerza.

En la figura vemos claramente que, ante un objeto de peso P [N], se utiliza una polea fija y una cuerda para levantarlo.

Llamaremos T a la tensión de la cuerda y P a la fuerza que se debe ejercer para levantar la carga.

Pero, a la vez, podemos apreciar que la polea modifica la dirección y el sentido de la fuerza que se necesita para levantar el cuerpo.

Siguiendo con la denominación utilizada en palancas para el caso de poleas fijas, la fuerza activa F_a que se debe realizar es igual a la fuerza de la carga F_c , por lo que –en este caso– no hay ningún provecho respecto a fuerzas, es decir $F_a = F_c$.

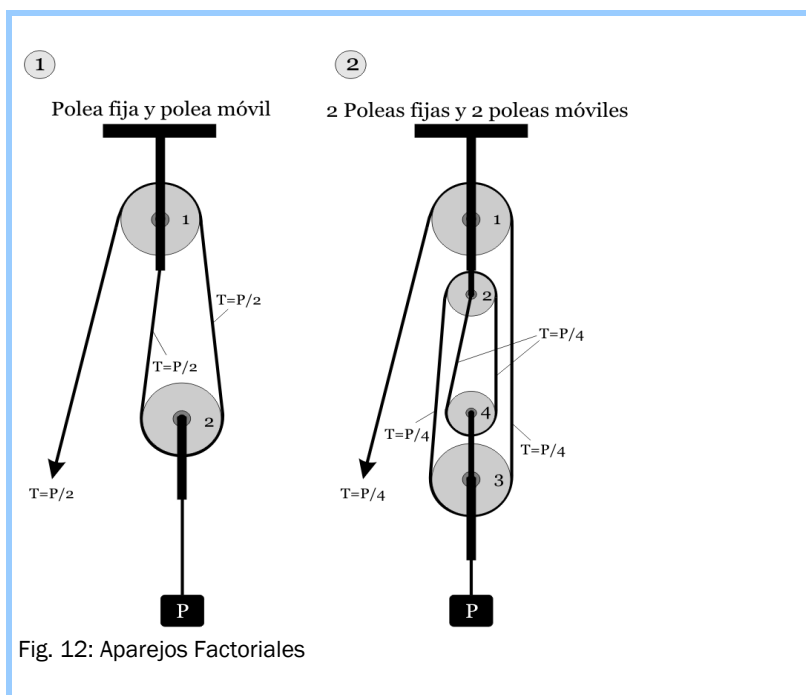


Poleas fijas Son aquellas que no cambian de posición y solamente giran alrededor de su eje.

Poleas móviles Son aquellas que, además de girar sobre su eje, se desplazan.

2.2.3. Aparejos

Al combinar poleas obtenemos el llamado *aparejo*, que se clasifica en función del tipo, de la cantidad y ubicación de las poleas.



En el caso 1 de la Fig. 12, el aparejo está formado por una polea fija y una polea móvil, y la fuerza activa F_a es la mitad del peso P .

En el caso 2, tenemos dos poleas fijas, que son las poleas 1 y 2, combinadas con dos poleas móviles (3 y 4). En este caso, la fuerza activa F_a es el $\frac{1}{4}$ del peso P que queremos levantar.

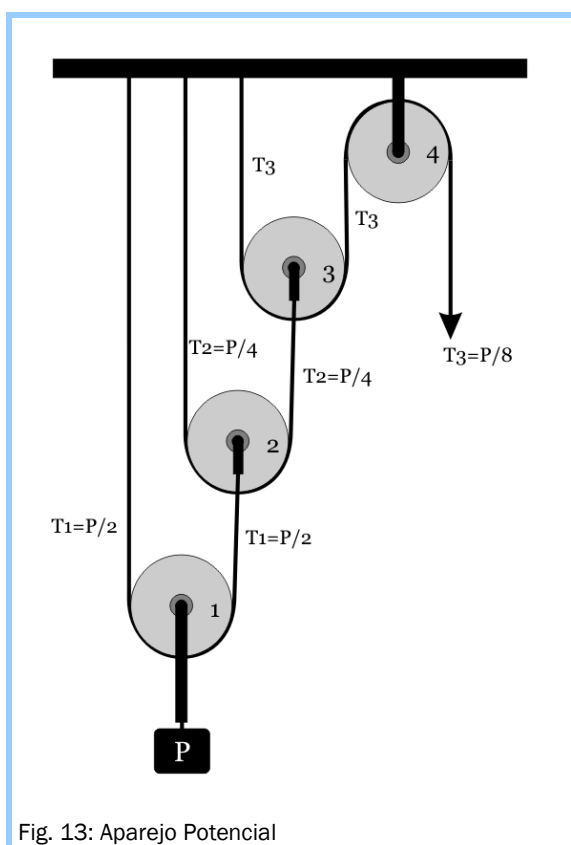
La expresión general para aparejos factoriales es:

$$F_a = \frac{F_c}{2 \cdot n}$$

Donde n = número de poleas móviles.

Aparejo potencial

Está constituido por una polea fija y varias poleas móviles.



Como vemos en la Fig. 13, la polea móvil 1 reduce la fuerza a P en dos mitades, llamadas cada una $T1$; entonces $T1 = P/2$.

Luego, la polea móvil 2 reduce en dos mitades a $T1$, llamadas cada una $T2 = T1/2$.

Después, la polea móvil 3 reduce a $T2$ en dos mitades, llamadas cada una $T3 = T2/2$, y finalmente, la polea fija 4 no reduce la fuerza $T3$, sino que le cambia el sentido.

Así, tenemos que la fuerza en el extremo de la sog a es $T3 = T2/2$

pero como $T2 = T1/2$

y a su vez $T1 = P/2$

nos queda que $T3=P/4$

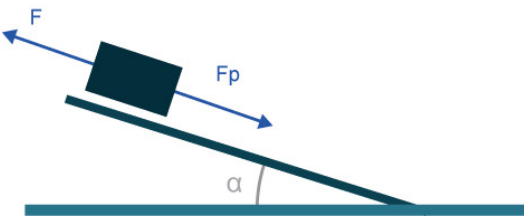
Podemos concluir, entonces, que la expresión general para aparejos potenciales es:

$$F_a = \frac{F_c}{2^n}$$

Donde n = número de poleas móviles.

2.3. Plano inclinado

Un plano inclinado es una superficie rígida que forma un ángulo con la horizontal.

 <p>Fig. 14</p>	<p>En la Fig. 14 puede apreciarse un plano inclinado y un objeto sobre él.</p> <p>Para que se cumpla la condición de equilibrio, la sumatoria de todas sus fuerzas debe ser cero.</p> <p>Por lo tanto $F - F_p = 0$</p> <p>F es la fuerza que se debe ejercer para arrastrar el objeto por la superficie y Fp es la componente del peso P del objeto en dirección al plano inclinado, y:</p> $F_p = P \cdot \text{sen} \alpha$ <p>Por lo tanto, la condición de equilibrio se da cuando:</p> $F - P \cdot \text{sen} \alpha = 0$ <p>Es decir: $F = P \cdot \text{sen} \alpha$</p>
--	---

2.4. Leyes del movimiento (Leyes de Newton)

Las tres leyes de Newton son conocidas como las leyes de movimiento de los cuerpos y se definen de la siguiente manera:

Primera Ley de Newton o Principio de Inercia Se denomina *inercia* a la resistencia que presenta un objeto a los cambios en su estado de movimiento. Esto significa que si un objeto está en reposo, tenderá a seguir en su estado de reposo, y si un objeto está moviéndose a velocidad constante, tenderá a seguir en ese mismo estado.

“Todo cuerpo continuará en su estado de reposo o de velocidad constante en línea recta (MRU) mientras sobre él no actúe una fuerza externa (neta) que lo haga cambiar su estado de movimiento”¹.

Segunda Ley de Newton o Principio de Masa Cuando un cuerpo es empujado, éste adquiere una aceleración en el mismo sentido que la fuerza aplicada. La rapidez que desarrolla el cuerpo aumenta si la fuerza aplicada tiene el mismo sentido que la velocidad, mientras que su rapidez disminuye si la fuerza se opone a la velocidad.

¹ Botto, J.; González, N.; Muñoz, J. Fís Física. Buenos Aires, Tinta Fresca, 2006.

“La aceleración de un cuerpo es directamente proporcional a la fuerza neta que actúe sobre él”².

Esta ley se representa con la siguiente ecuación: $\vec{a} = \vec{F} / m$

Despejando la ecuación anterior, queda: $\vec{F} = a.m$.

F= fuerza externa

M= masa inercial

Si las reemplazamos por sus unidades, llegamos a las unidades que forman el Newton:

$$[N] = [m/s^2] \cdot Kg$$

Tercera Ley de Newton o Principio de Interacción En nuestro universo existen fuerzas de interacción. Por ello el sol atrae a la tierra y viceversa.

Este principio, también conocido como *Acción y Reacción*, sostiene que:

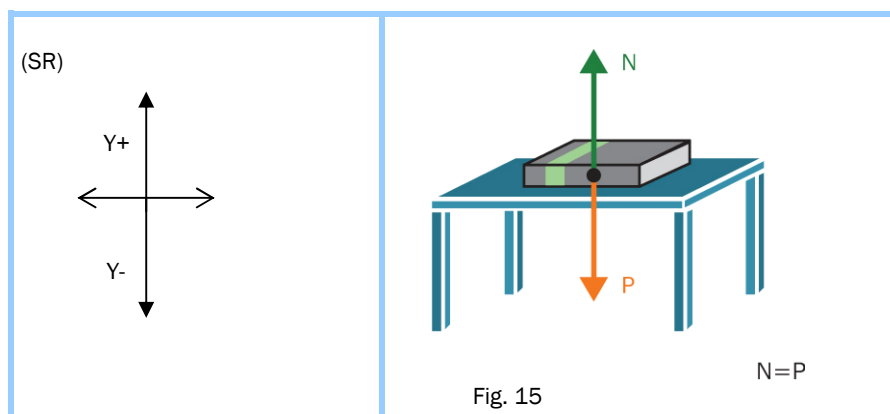
“Siempre que un cuerpo A ejerza una fuerza sobre un cuerpo B, entonces el cuerpo B también ejerce una fuerza sobre el cuerpo A, de igual magnitud pero en sentido contrario a la primera”³.

Las fuerzas provenientes de la interacción entre dos cuerpos siempre actúan sobre objetos *diferentes*. Son un par de interacción. De ahí el nombre de este principio. Para explicar esto con un ejemplo, supongamos que, si empujamos la pared hacia adentro con una mano, recibiremos desde ella una fuerza hacia fuera, de igual magnitud pero de sentido contrario. La pared es rígida y más resistente que la mano, y por ello sentimos dolor en ella.

2.5. Diagrama de cuerpo libre

Se denomina así a la representación gráfica de *todas las fuerzas* que actúan sobre un cuerpo, al cual se realiza el estudio del movimiento o de su estado de equilibrio.

Para nuestro análisis, hagamos de cuenta que tenemos un libro en reposo, apoyado arriba de una mesa:



² Botto, J.; González, N.; Muñoz, J, op. cit.

³ Botto, J.; González, N.; Muñoz, J., op. cit.

Actúan dos fuerzas: su propio *Peso* (P), que siempre se dibuja desde el centro del objeto y hacia abajo, y la acción que ejerce la mesa para sostener al libro. La misma se denomina *fuerza Normal* (N) o fuerza de vínculo.

Esta fuerza actúa siempre *perpendicularmente* a la superficie de apoyo.

En este caso, como la superficie de apoyo es horizontal, el valor de la fuerza Normal es igual al *Peso* del cuerpo. Por lo tanto:

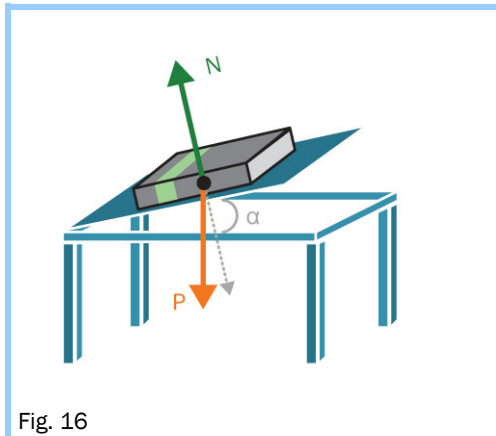
$P = N$, tal como muestra la Fig. 15.

2.5.1. Sistema de referencia

Es sumamente necesario fijar un sistema de referencia (SR). En el caso de nuestro ejemplo, es fácil notar que todo se desarrolla en el eje Y , donde la fuerza Normal posee un valor positivo (se orienta hacia el eje positivo de las Y) y la fuerza *Peso* es negativa, ya que su sentido es hacia el eje negativo del eje de las ordenadas.

¿Seguimos analizando?

¿Qué pasa, ahora, si al mismo libro lo ubicamos sobre un plano inclinado?



La fuerza *Peso* siempre actúa desde el centro del cuerpo y verticalmente hacia abajo, mientras que la fuerza Normal (como ya hemos dicho) siempre actúa perpendicularmente a la superficie de apoyo. Por lo tanto, ambas fuerzas no son iguales, tienen distinto valor. Entonces:

$$P \neq N$$

Recuerden esto, ya que más adelante será demostrado.

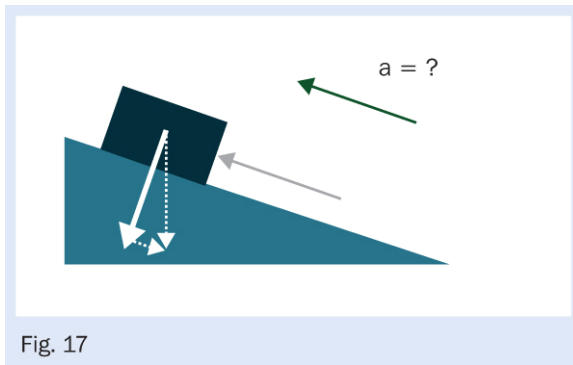
¿Cómo resolvemos esto?

Esta es la rama de la Física y la Mecánica, denominada *dinámica de la partícula*, que estudia el movimiento de los cuerpos, interesándose en las causas que lo producen.

En nuestro ejemplo del libro, éste no se mueve, ya que sigue en su estado de equilibrio, pero para que lo haga, necesita una fuerza externa que lo empuje y venza su estado de inercia.

Dibujemos otra vez el libro sobre el plano inclinado y un ángulo α . Para que el objeto suba por el plano debemos aplicar una *fuerza externa*, tal como muestra la Fig. 17 de diagrama de cuerpo libre.

Si el objetivo de este análisis es calcular con qué *aceleración* sube el cuerpo, no olvidemos que al aplicar una fuerza, el objeto se acelera.



En el diagrama de cuerpo libre se dibujan todos los vectores y, como ya hemos visto, se trabaja sumando o restando las componentes vectoriales de cada uno de ellos, presentes en cada eje cartesiano.

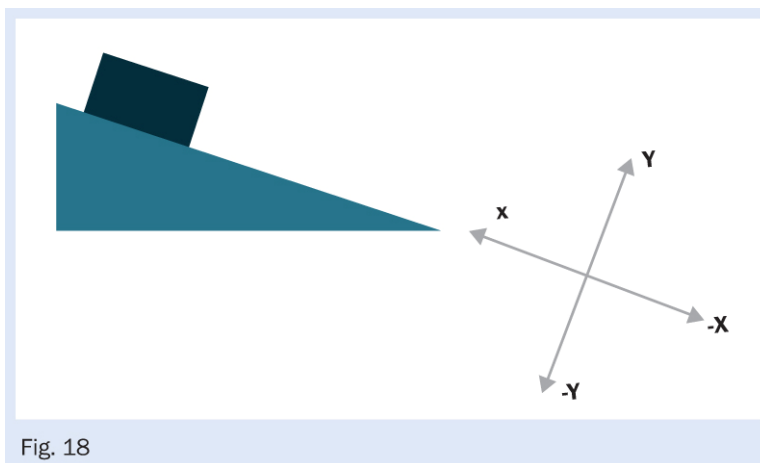
¿Cómo vamos a trabajar?

Retomemos el ejemplo del libro, pero utilizando valores reales. Supongamos que el libro posee una masa de 2 Kg y es empujado hacia arriba sobre un plano inclinado 30° con la horizontal, con una fuerza de 20 [N]. ¿Cuál es la aceleración que adquiere el libro al subir?

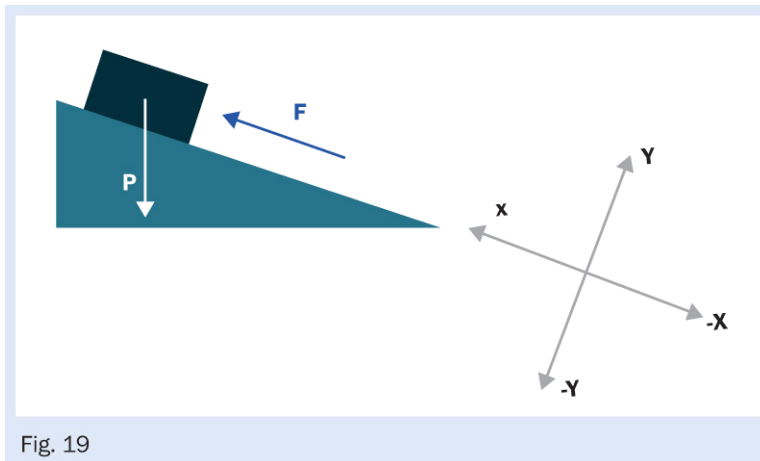
1. Primero, es necesario colocar un sistema de referencia (SR). Recordemos que el mismo es un sistema de ejes cartesianos, que son perpendiculares entre sí y forman cuatro ángulos rectos.

Para nuestro ejemplo, el SR se logra de la siguiente manera:

Se continúa la recta de la hipotenusa del plano inclinado y luego se dibuja una perpendicular a la misma:

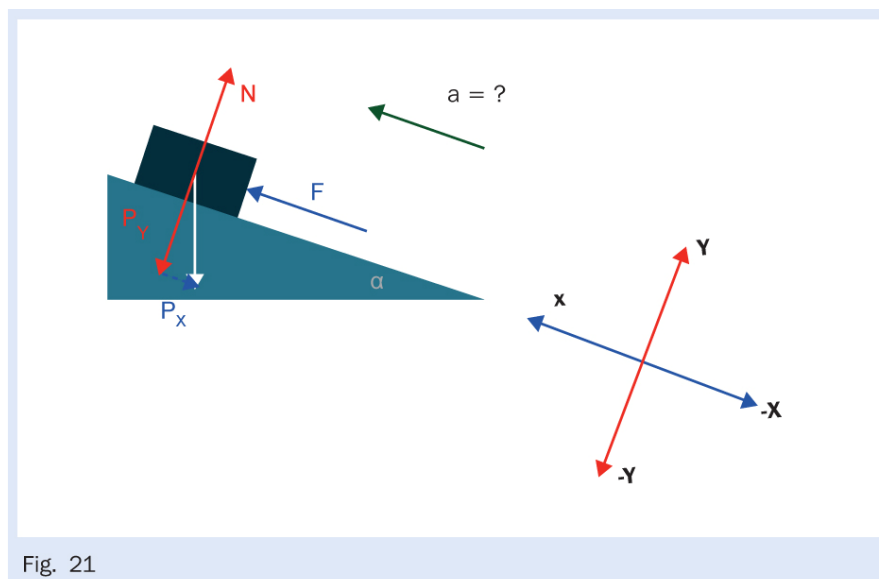
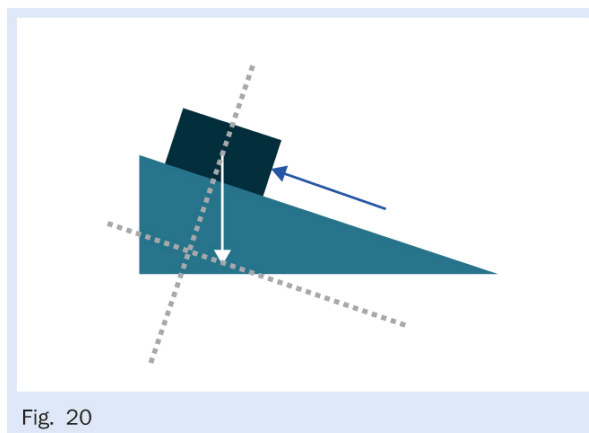


2. Se completa el diagrama de cuerpo libre con todos los vectores. Sólo actúan la fuerza F que empuja y el peso P del cuerpo:



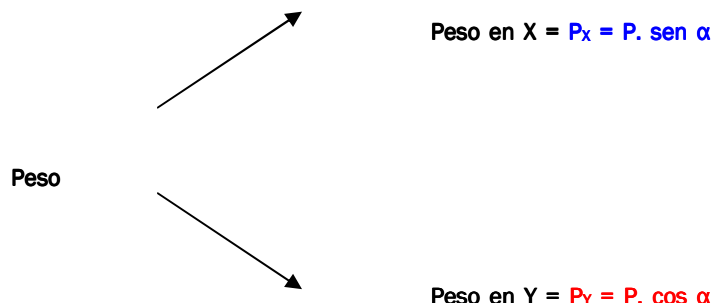
Para poder sumar los vectores, éstos deben estar –como mínimo– *orientados* a cada uno de los ejes cartesianos. Si observamos la Fig. 19, vemos que la fuerza F sí está orientada al eje X , pero la del Peso, no. Por tal motivo, tendremos que desagregar esta fuerza en sus componentes vectoriales de la siguiente manera:

3. Desde donde termina la fuerza Peso (la punta de flecha), trazamos la paralela a la hipotenusa del plano inclinado; y desde donde comienza el Peso (centro del cuerpo), la perpendicular a la paralela anterior, logrando así las componentes del Peso en X y del peso en Y :



4. En el esquema anterior hemos diferenciado a propósito cada eje, asignándoles colores distintos: el **color azul** para el eje X y el **color rojo** para el eje Y. De la misma manera, cada fuerza actuante sobre el cuerpo está distinguida con el color del eje al cual pertenece.

El peso del cuerpo se descompone en Peso en X (P_x) y Peso en Y (P_y). Sumando estos dos vectores, por el método de la poligonal (visto en la unidad anterior), obtenemos el vector Peso (dibujado con color negro en el esquema):



2.5.2. Cálculo de la aceleración

Para trabajar correctamente en este tipo de problemas, debemos comenzar con las sumatorias de las distintas fuerzas en cada eje, *aplicando la Segunda Ley de Newton*, de la siguiente forma:

Sumatoria en el eje Y: $\Sigma \text{Fuerzas}_{\text{eje Y}} = m \cdot a_y$

Sumatoria en el eje X: $\Sigma \text{Fuerzas}_{\text{eje X}} = m \cdot a_x$

Siendo a_y la aceleración en el eje Y, a_x la aceleración en el eje X.

5.a. Entonces:

$$\Sigma \text{Fuerzas}_{\text{eje Y}} = m \cdot a_y$$

$$N - P_y = m \cdot a_y \quad +$$

Recordemos que todo lo correspondiente al eje Y está en color rojo.

$$N - P \cdot \cos \alpha = m \cdot a_y$$

Además, de acuerdo al SR, la N es positiva (orientada hacia el eje positivo de las Y) y el peso en Y está hacia el eje Y negativo.

Analizando el movimiento del cuerpo, denotamos que *no hay movimiento sobre el eje Y*, sino que sólo se desplaza sobre el eje x. Por tal motivo, *no hay aceleración en el eje Y*, con lo cual a_y es igual a cero: $a_y = 0$.

Entonces:

$$N - P \cdot \cos \alpha = m \cdot a_y = 0$$

$$N = P \cdot \cos \alpha$$

$$N = m \cdot g \cdot \cos \alpha$$

$$N = 2 \text{ kg} \cdot 9,8 \text{ m/s}^2 \cdot \cos 30^\circ$$

$$N = 17 \text{ [N]}$$

Valor de la fuerza Normal

¿Recuerdan que cuando analizamos el ejemplo del libro, una páginas más arriba, habíamos concluido que $P \neq N$? Lo que acabamos de hacer es explicar lo que sucede con el Peso de un cuerpo apoyado sobre un plano inclinado.

5. b– Siguiendo el análisis

$$\Sigma \text{Fuerzas}_{\text{eje } x} = m \cdot a_x$$

En color azul denotamos todos los vectores en el eje X.

$$F - P_x = m \cdot a_x$$

De acuerdo al SR, la F que empuja es positiva y el peso en X está hacia la derecha, orientado hacia el eje negativo de las X.

$$F - P \cdot \text{sen} \alpha = m \cdot a_x$$

$$(F - P \cdot \text{sen} \alpha) / m = a_x$$

$$(20 \text{ [N]} - 2 \text{ kg} \cdot 9,8 \text{ m/s}^2 \cdot \text{sen } 30^\circ) / 2 \text{ kg} = 5,1 \text{ m/s}^2$$

$$a_x = 5,1 \text{ m/s}^2$$

Ahora bien, ¿ésta es la aceleración que adquiere el libro al subir por el plano inclinado?!

¡No todo es tan sencillo ni todo tan fácil!

En la vida real esto no sucede; se comprueba experimentalmente que el libro *no sube* con dicha aceleración, la cual es menor a ese valor. ¿Por qué? Aparece una fuerza que siempre está y se opone al movimiento; la llamada

fuerza de Rozamiento o fuerza de fricción (F_R).

2.5.3. Fuerza de Rozamiento

Siempre está presente entre *dos superficies en contacto* y expresa la resistencia que presentan las mismas al desplazarse una con respecto a la otra.

Fuerza de Rozamiento estática

Es la responsable de impedir el desplazamiento de una superficie con respecto a la otra.

Fuerza de Rozamiento dinámica

Es la fuerza de interacción entre las dos superficies.

En ambos casos, la fuerza de rozamiento depende de la fuerza *Normal* y de un coeficiente de rozamiento μ . Éste, a su vez, depende de los materiales de cada superficie en contacto y es *adimensional* (no posee unidades).

Matemáticamente, esto se expresa así:

$$F_{R\text{e}} = \mu_{\text{e}} \cdot N \quad \text{Fuerza de rozamiento estática}$$

$$F_{R\text{d}} = \mu_{\text{d}} \cdot N \quad \text{Fuerza de rozamiento dinámica}$$

Para tener en cuenta

$0 < \mu_d < 1$ significa que el valor de dicho coeficiente dinámico está entre 0 y 1 y no tiene unidades.

2.5.4. Coeficientes de rozamiento

Superficies en contacto	Coeficiente de rozamiento
hielo sobre hielo	0,02
cuero sobre madera	0,3 - 0,4
cuero sobre metal	0,6
acero sobre acero	0,6
madera sobre madera	0,25 - 0,50
latón sobre hielo	0,02

Entonces, volviendo a nuestro ejemplo del libro que subía por el plano inclinado, por qué habíamos calculado erróneamente la aceleración con la que subía? Porque no tuvimos en cuenta la *fuerza de rozamiento*, que siempre se opone al movimiento.

Por lo tanto, en el esquema de cuerpo libre, en el eje donde se produce el movimiento (eje X), dibujaremos de la siguiente manera:

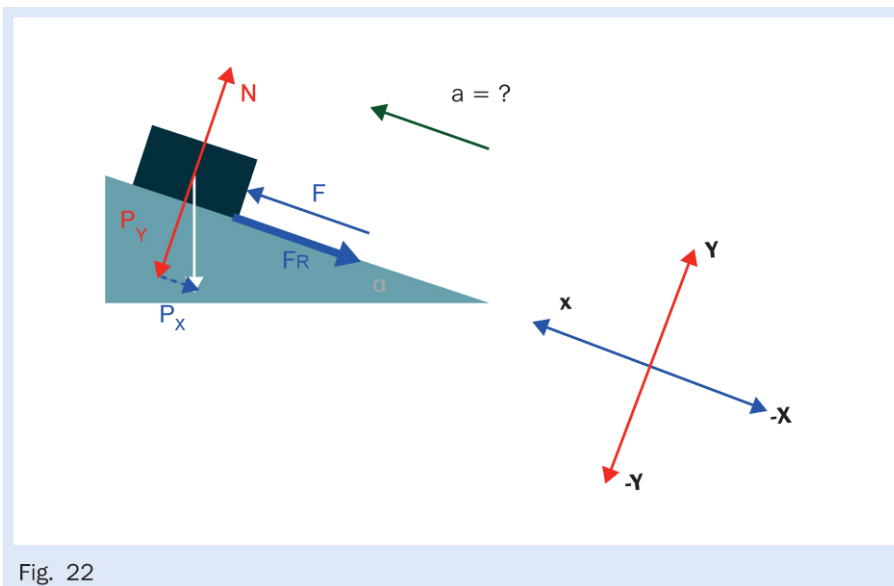


Fig. 22

- En el eje Y continúan los dos vectores dibujados con color rojo, la N y el P_y , y es válido el cálculo anterior:

$$N = 2 \text{ kg} \cdot 9,8 \text{ m/s}^2 \cdot \cos 30^\circ$$

$$N = 17 \text{ [N]}$$

Valor de la fuerza Normal

- En cambio, en el eje X, la situación ahora cambió, ya que –si observamos atentamente– es posible notar tres vectores dibujados con color azul, la F , el P_x y la F_R . Entonces, la situación será la siguiente:

$$\Sigma \text{Fuerzas}_{\text{ejeX}} = m \cdot a_x$$

$$F - P_x - F_R = m \cdot a_x$$

De acuerdo al SR, la F que empuja es positiva y el peso en X está hacia la derecha, orientado hacia el eje negativo de las X, como también la fuerza de rozamiento.

$$F - P \cdot \text{sen}\alpha - \mu_d \cdot N = m \cdot a_x \quad \text{El } \mu_d \text{ entre las dos superficies es de 0,3.}$$

$$(F - P \cdot \text{sen}\alpha - \mu_d \cdot N) / m = a_x$$

$$(20 \text{ [N]} - 2 \text{ kg} \cdot 9,8 \text{ m/s}^2 \cdot \text{sen } 30^\circ - 0,3 \cdot 17 \text{ [N]}) / 2 \text{ kg} = 2,55 \text{ m/s}^2$$

$$a_x = 2,55 \text{ m/s}^2$$

¡Ahora sí podemos decir que ésta es la aceleración con la que sube el libro!

2.6. Tipos de fuerzas

A lo largo de esta unidad hemos estudiado distintas fuerzas como Peso, Normal, de Rozamiento.

Sin embargo, a esta lista podemos agregar otras muy importantes como las fuerzas de atracción electrostáticas; fuerzas magnéticas; fuerzas de atracción gravitatoria; fuerzas nucleares débiles, y una muy útil para nuestros análisis de eventos físicos, que es la fuerza elástica.

2.6.1. Fuerza Elástica

Antes de definir esta fuerza, debemos saber a qué se denomina *elasticidad* de un cuerpo, y ésta es la propiedad que tiene el mismo para *modificar su forma*, cuando se le aplica una fuerza, y de recuperarla, cuando dicha fuerza cesa su acción.

El ejemplo más comúnmente citado es el resorte (elástico), el cual tiene la propiedad de alargarse o comprimirse, según la fuerza aplicada. Pero no nos equivoquemos al pensar que sólo los resortes poseen dicha propiedad, porque también hay otros objetos elásticos, como la pelota, la cual, al caer y golpear contra el suelo o al ser pateada, se deforma y luego vuelve a recuperar inmediatamente su fisonomía original. Sólo si el valor de la fuerza aplicada es excesivamente grande, el objeto se deforma y no recupera su forma original.

Para comprender mejor las características de esta fuerza, supongamos que tenemos un resorte del cual se cuelga un cierto Peso (P_1) en uno de sus extremos. Lógicamente, ese resorte se estirará un poquito.

Ahora bien, si sacamos ese P_1 y colgamos el *doble* de peso (P_2), el resorte se estirará el *doble* (longitud de estiramiento). Y si finalmente colgamos el triple de peso de P_1 , la longitud de estiramiento también será el triple de la primera.

Con este ejemplo podemos concluir que la *elongación* o longitud del estiramiento es *directamente proporcional a la fuerza* (Peso que se cuelga). Dicha proporcionalidad directa se denomina K y es la llamada *constante elástica de un resorte*. Esta constante es una propiedad que tiene cada resorte y depende del material del cual está hecho, el diámetro, etc., y hay valores de K que están tabulados para distintos resortes.

La expresión matemática de esta fuerza elástica es:

$$F_e = K \cdot \Delta X$$

F_e = Fuerza elástica y su unidad es [N] (Newton)

ΔX = Elongación o compresión del resorte, se mide en [m]

K = Constante elástica del resorte cuyas unidades son [N / m]

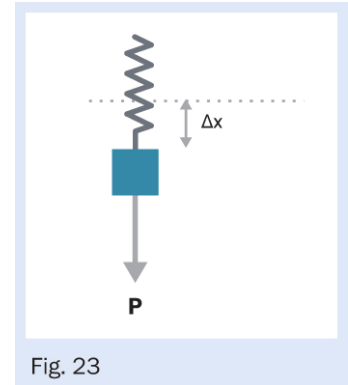


Fig. 23

2.7. Motor de Física

2.7.1. ¿Qué es Box2D?

Box2D es un motor de física que, como su nombre lo indica, trabaja solamente con geometría en 2D. En particular, el motor utiliza detección de colisión continua o discreta y simulación de cuerpo rígido. Las colisiones son resueltas con un esquema iterativo, utilizando impulsos. Además, está hecho en C++ y utiliza el paradigma de orientación a objetos.

¡No se asusten! Estos conceptos, desconocidos para muchos de ustedes, serán dados a lo largo de la materia. Mientras tanto, en esta unidad les brindaremos nociones básicas sobre Box2D y cómo empezar a utilizarlo para simular objetos sujetos a fuerzas simples, tal como hemos visto en esta unidad.

¿Por qué Box2D y no otro motor?

Elegimos usar Box2D y no otro motor de física por dos motivos:

- Este motor se hizo a modo de ejemplo o tutorial para verificar ciertas técnicas de simulación física y detección de colisiones. Por lo tanto, es relativamente simple...
- No obstante el punto anterior, Box2D es un muy buen motor físico que demostró su estabilidad y rendimiento en varias aplicaciones comerciales y no comerciales.
- Existen ports de Box2D para otros lenguajes/plataformas (c#, flash, etc.).

Para tener en cuenta

Para usar el código, primero debemos tener en cuenta ciertas particularidades de Box2D:

- Absolutamente todas las clases de Box2D empiezan con el prefijo b2. Entonces, por ejemplo, tendremos las clases: b2World, b2Body, b2Fixture, etc.
- Todas las clases b2Body, b2Fixture, b2Joint y b2Constraint y sus respectivas clases hijas son creadas por medio de *factories* (luego veremos qué quiere decir esto). Esto implica una suerte de *juramento* que siempre deberemos respetar si usamos Box2D:
 - Jamás instanciaré una clase de tipo b2Body, b2Fixture, b2Joint, b2Constraint o clases derivadas, de forma directa (es decir, con un new), sino que siempre lo haré mediante la clase b2World o b2Body, según corresponda.
 - Jamás borraré una de estas clases de forma directa (con delete), sino que siempre lo haré mediante la clase b2World o b2Body, según corresponda.

Dicen que al que no cumple con este juramento, le recaerá la maldición de *NULL exception* o los comportamientos impredecibles...

2.7.2. Factories

Box2D utiliza un patrón de creación de objetos conocido como *fábrica* o *factory*. El mismo consiste en utilizar una clase, denominada *constructora*, que se encarga de instanciar objetos, llamados *productos*. Esto implica que cuando necesitamos crear alguno de esos objetos producto, no lo hacemos mediante un *new*, como es tradicional en C++, sino que invocamos a la clase constructora, la cual se encarga de crear el objeto y devolvernos un puntero al mismo.

La clase constructora encapsula toda la lógica y el conocimiento necesarios para inicializar un objeto producto y para destruirlo, por lo que tampoco nosotros liberaremos la memoria de los objetos, sino que le pediremos a la clase constructora que lo haga.

A continuación, se enuncia un ejemplo sencillo de este patrón:

```
class Enemigo{
    //...
    //contenido de la clase
    //...
};

class Constructora{
public:
    Enemigo* ConstruirEnemigo(){
        Enemigo *enemigo= new Enemigo();
        //operaciones necesarias para
        configurar el enemigo
        return enemigo;
    }

    void DestruirEnemigo(Enemigo *en){
        //tareas necesarias para liberar
        //al enemigo
        delete en;
    }
};

void main()
{
    Constructora *C= new Constructora();
    Enemigo* enemigo1= C->ConstruirEnemigo();
    //realizar tareas con el objeto
    //y luego para liberarlo hacemos
    C->DestruirEnemigo(enemigo1);
}
```

Factories

Es un patrón de creación de objetos utilizado por Box2D. Consiste en utilizar una clase, denominada constructora, que se encarga de instanciar objetos, llamados productos.

En este ejemplo se declara una clase *Constructora* y otra *Enemigo*. *Constructora* es la responsable de crear y destruir los objetos de tipo *Enemigo*, como se muestra en la rutina *main*.

2.8. Entonces, ¿cómo se usa?

2.8.1. El mundo de la simulación

Toda simulación física en Box2D depende de la clase *b2World*. Ésta representa el mundo físico donde se realizará la simulación es decir, donde vivirán los cuerpos,

restricciones (constraints) y uniones. La clase es la responsable de crear todas las entidades físicas, realizar la simulación y responder a consultas sobre la escena.

Nuestro juego puede tener tantas instancias de esta clase como deseemos, pero lo común es tener sólo una. Si bien podríamos tener varias, cada una de ellas llevaría la simulación de un escenario físico, pero de forma separada, y no existiría ninguna relación entre las simulaciones, lo cual -en muchos casos- no nos serviría.

La instanciación de la clase tiene sólo un constructor, que está definido de la siguiente forma:

```
b2World(const b2Vec2& gravity, bool doSleep);
```

Los parámetros son:

- El vector de gravedad que se aplicará a los cuerpos en la simulación (queda implícito que todos los cuerpos tendrán el mismo vector de gravedad).
- doSleep es una bandera que le dice si el motor puede optimizar la simulación mediante una característica que veremos más adelante.

Para instanciar esta clase lo podemos hacer en el *stack* o en el *heap* (mediante *new*). Entonces podemos hacer:

```
//Creamos el mundo de la simulación en el stack
B2World m_World(gravity, doSleep);

//Creamos el mundo de la simulación en el heap
B2World* pWorld = new B2World (gravity, doSleep);
```

En el caso de que usemos el *heap* para instanciar la variable, recordemos que deberemos borrar la variable usando el operador *delete*:

```
delete pWorld;
```

En el caso de que se almacene en el *stack*, recordemos que la variable se eliminará automáticamente al salir del *scope* en que fue creado.

Esta clase *b2World* es la que funcionará como *factory* para los objetos de simulación que veremos a continuación.

2.8.2. Objetos de simulación

Para instanciar las entidades que queremos simular -las que serán los enemigos, nuestros aliados, nuestros edificios e incluso posiblemente nuestro personaje principal- primero deberemos crear una definición para estar las mismas.

El hecho de tener que crear una definición antes de instanciar las entidades físicas se debe a que, como dijimos antes, la instanciación propiamente dicha la hará la clase *b2World*. Nosotros le solicitaremos la instanciación y le suministraremos una definición para el objeto que queremos crear. Para ser más concretos, las entidades de las que hablamos son *b2Body*, *b2Fixture*, *b2Joint* y *b2Constraint* y sus respectivas clases hijas. Veamos, a continuación, un par de éstas.

B2Body: cuerpos

Son los cuerpos rígidos de la simulación. Cualquier objeto dinámico, cinemático o estático que queramos en la simulación necesitará que lo definamos como un *b2Body* y esto se logra usando la estructura *b2BodyDef*, la cual posee los siguientes campos:

- *b2BodyType type*: puede ser *static*, *kinematic* o *dynamic*.

- Un objeto *static* o *estático* es aquel que será inamovible en nuestra simulación, o mejor, que sólo será movable por código, pero no dentro de la simulación. Estos objetos tienen masa 0 (por convención, su masa, en realidad, simula la de uno infinito), poseen velocidad 0 y sólo pueden ser movidos por el programador. Entre otros ejemplos de objetos estáticos en una escena, podemos mencionar pisos, paredes, puertas, edificios, etc.
 - Un objeto *kinematic* o *cinemático* es aquel que es movido por velocidades y/o aceleraciones, pero no por fuerzas. Estos objetos tienen masa distinta de 0, poseen velocidad seteada por el programador y son movidos por el integrador de Box2D. Ejemplos de objetos cinemáticos en una escena son elementos del GUI o algunos tipos de enemigos que no requieran de fuerzas para su movimiento.
 - Un objeto *dynamic* o *dinámico* es aquel que es movido por fuerzas, es decir, que se comporta como en el mundo real. Estos objetos tienen masa distinta de 0, poseen velocidad determinada por la fuerza y son movidos por el integrador de Box2D. Ejemplos de objetos dinámicos en una escena son rocas y otros elementos como disparos.
- *b2Vec2 position*: es la posición en el mundo del cuerpo.
 - *float32 angle*: es el ángulo del objeto en radianes.
 - *b2Vec2 linearVelocity*: es la velocidad inicial del centro del cuerpo en coordenadas del mundo.
 - *float32 angularVelocity*: es la velocidad angular del objeto.
 - *float32 linearDamping*: es la reducción de la velocidad lineal del objeto.
 - *float32 angularDamping*: es la reducción de la velocidad angular.
 - *bool fixedRotation*: si el objeto es verdadero, no rotará, pero si es falso, rotará de forma natural. Esta condición sirve para los personajes que, en general, no queremos que roten o queden *caídos* en el suelo.
 - *bool allowSleep*: nos indica si el objeto puede ponerse en estado *sleep*.
 - *bool awake*: nos indica si este objeto está inicialmente despierto o dormido.
 - *bool bullet*: si es verdadero, considerará al objeto como uno que se mueve a gran velocidad (como es el caso típico de las balas; de ahí el nombre). Entonces empleará métodos continuos en la detección de colisiones para evitar el *tunneling*, es decir, que el objeto traspase a otros. El lado negativo de este campo es que si lo activamos, el cálculo será computacionalmente más costoso.
 - *bool active*: nos indica si empieza *active* el objeto.
 - *void* userData*: es un puntero que podemos usar para almacenar en el cuerpo una referencia a un objeto de nuestro interés, por ejemplo, el objeto que posee ese cuerpo.

Si bien, como podemos apreciar, son muchísimas propiedades, la creación no es tan exhaustiva y, en general, sólo establecemos un par de ellas dejando el resto por defecto, ya que son útiles para situaciones particulares.

Dijimos que todas las entidades físicas de la simulación deben ser creadas por *b2World* (claro está, a petición nuestra). Entonces, si deseamos crear un cuerpo para la simulación, lo haremos de la siguiente forma:

```
b2Body*      pCuerpo = NULL;
b2BodyDef    CuerpoDefinicion;
CuerpoDefinicion.type           =
b2BodyType::b2_dynamicBody;
CuerpoDefinicion.position = b2Vec2(100, 200);
pCuerpo = World.CreateBody(&CuerpoDefinicion);
```

Aquí debemos recordar el juramento que hicimos acerca de que no destruiremos ninguna variable de tipo *b2Body*, ya que eso lo debe hacer el *b2World* que lo creó. Por lo tanto, jamás haremos lo siguiente:

```
delete pCuerpo; // JAMAS DEBEMOS HACER ESTO
```

La forma correcta para liberarlo será:

```
World.DestroyBody(pCuerpo);
```

La clase `b2Body` no contiene información sobre las figuras de colisión ni la masa del objeto. En cambio, ésta contendrá varias figuras de colisión, cada una con su masa de manera individual, como veremos a continuación.

b2Fixture: adorno

En el punto anterior creamos cuerpos, pero ¿cómo eran? ¿Rectangulares, circulares, poligonales? ¿Y su masa? ¿Cuál era su masa?

Por lo tanto, ¿cómo podemos simular el cuerpo rígido, anteriormente creado? Si no sabemos siquiera su forma, no podemos colisionarlo... ¿Cómo lo afectarán las fuerzas? Si desconocemos su masa, no podemos usar las leyes de Newton... Además, otras propiedades del objeto son desconocidas, como su restitución (en otras unidades veremos de qué se trata esto).

Todos estos *huecos* en la definición del cuerpo rígido son establecidos por los adornos de Box2D. Su creación es bastante similar a la de los cuerpos. Esto significa que primero instanciaremos la estructura `b2FixtureDef` y luego, en base a la misma, setearemos el fixture al cuerpo que queramos.

Veamos, ahora, los campos de la definición:

- `const b2Shape* shape`: es un objeto que veremos a continuación, que indica cómo es la forma física del fixture, es decir, si es un círculo, un rectángulo, un polígono.
- `float32 friction`: es el coeficiente de fricción del cuerpo. En general, usaremos valores en el rango [0,1].
- `float32 restitution`: nos permite establecer el coeficiente de restitución del objeto. Por el momento tengamos en cuenta que, generalmente, es un valor en el rango [0,1].
- `float32 density`: indica la densidad del objeto. Es decir que, en lugar de establecer la masa del objeto, directamente le daremos la densidad de éste y, en base al área de la figura que posea, calculará automáticamente la masa. En general, las unidades de densidad se la pasaremos en kg/m^2 .
- `b2Filter filter`: esta es una característica muy buena que nos permite poner filtros a las colisiones de ciertos objetos. No explicaremos esto aquí, pero, por ejemplo, si nuestro juego transcurre en dos capas -una de fondo y otra de frente-, los objetos de la capa del fondo colisionarán sólo entre ellos, al igual que los del frente.
- `void* userData`: esta propiedad es análoga a la de `b2Body`.

Al igual que en `b2BodyDef`, muchas de esas propiedades no las cambiaremos.

Siguiendo el ejemplo de `b2Body`, crearemos un fixture para ese cuerpo:

```
//creamos la definición para nuestro fixture
b2FixtureDef AdornoDefinicion;

//le establecemos una friccion que nos guste
AdornoDefinicion.friction = 0.3f;
//y una densidad
AdornoDefinicion.density = 1.0f;
```

En este caso, nos faltó especificar la propiedad `shape` que le dirá al fixture cuál es su forma física, para el cálculo de las colisiones, el momento de inercia y su masa, en función de la densidad que le pasamos.

Las formas que podemos crear serán:

- `b2CircleShape` para círculos

- b2PolygonShape para polígonos

Un b2CircleShape tendrá la propiedad *radius* que indicará el radio del círculo. Un ejemplo de creación de una forma de círculo es:

```
b2CircleShape pelota;
//le pasamos el radio del círculo
pelota.m_radius= radius;
```

Por otro lado, b2PolygonShape nos permitirá crear polígonos regulares e irregulares. Un ejemplo común son los rectángulos o cuadrados. Un ejemplo de creación de un rectángulo es:

```
b2PolygonShape floor;
//notemos que los valores que se le pasa son la
mitad del ancho y
//la mitad del alto respectivamente
floor.SetAsBox(200, 100);
```

Una vez que tenemos un b2Shape, podemos asignárselo al fixture y, de una vez por todas, instanciarlo. Concluamos, entonces, con el ejemplo de la creación del fixture:

```
b2CircleShape pelota;
pelota.m_radius = radius;
//establecemos la forma al fixture
AdornoDefinicion.shape = &pelota;

//le decimos al cuerpo rígido que instance el
fixture y se lo establezca
b2Fixture* pAdorno = pCuerpo-
>CreateFixture(&AdornoDefinicion);
```

Como hemos visto anteriormente, aquí es posible notar que al fixture lo instancia una factory que, en este caso, es una instancia de la clase b2Body, a la cual queremos asignar el fixture.

Un cuerpo sin fixtures no puede colisionar con nada en el mundo ni simularse.

2.9. Armandó el rompecabezas

2.9.1. Cómo avanzar en el tiempo

Una vez que tenemos la escena con los cuerpos creados, pasamos a avanzar la simulación en cada *frame* del juego. A propósito, hemos dicho que la simulación en su totalidad es realizada por la clase b2World, así que para avanzar en el tiempo, usaremos el método:

```
void Step(          float32 timeStep,
                  int32 velocityIterations,
                  int32 positionIterations);
```

- El tiempo para avanzar en la simulación.
- Recordemos que Box2D usa un esquema iterativo para la resolución de constraint. Aquí le pasamos la cantidad de iteraciones para las constraint de velocidad.
- Al igual que en el parámetro anterior, el método Step recibe la cantidad de iteraciones, pero para las constraint de posición.

La función se puede usar de la siguiente manera:


```
//Avanzamos a un paso de 1/60 segundos, ya que
tendremos 60 frames por segundo en nuestro juego

//Usaremos 10 iteraciones para constraint de
velocidad y posicion

World.Step(1/60.0f, 10, 10);
```

2.9.2. La simulación

Para aplicar todo lo que hemos aprendido hasta ahora, realicemos un pequeño ejemplo de simulación .

La misma constará de una pelota que cae y rebota sobre el suelo.

```
B2World World ( b2Vec2(0, -9.8), true) ;
B2BodyDef DefinicionCuerpo;
b2FixtureDef AdornoDefinicion;

//Creamos la pelota
DefinicionCuerpo.type = b2BodyType::b2_dynamicBody;
CuerpoDefinicion.position = b2Vec2(0, 100);

AdornoDefinicion.restitution = 1.0f;
AdornoDefinicion.density = 1.0f;
b2CircleShape pelota;
pelota.m_radius = radius;
AdornoDefinicion.shape = &pelota;

pCuerpo = m_World.CreateBody (&CuerpoDefinicion);
pAdorno = pCuerpo->CreateFixture(&AdornoDefinicion);
```

```
//Creamos el suelo
DefinicionCuerpo.type = b2BodyType::b2_staticBody;
CuerpoDefinicion.position = b2Vec2(0, 0);
AdornoDefinicion.density = 0.0f;
b2PolygonShape floor;
floor.SetAsBox(width/2.0f, 100.0f);
AdornoDefinicion.shape = &floor;

pCuerpo = m_World.CreateBody (&CuerpoDefinicion);
pAdorno = pCuerpo->CreateFixture(&AdornoDefinicion);

while(simulacion)
{
    World.Step(1/60.0f, 10, 10);
    m_World.ClearForces();
}
```

Recordemos que si bien realizará todos los cálculos de la simulación, Box2D no es una librería grafica. Por lo tanto, no tendremos ninguna forma gráfica para comprobar los resultados. Para esto, podemos hacer salidas por consola, a fin de conocer la posición del objeto de la pelota, o bien implementar la parte grafica de la simulación usando SFML.

Veremos así que la metodología es muy versátil. Box2D nos realizará toda la simulación, en tanto que podremos hacerla salida gráfica usando cualquier método (SFML, OpenGL, DirectX, Alegro, etc.).

2.9.3. Interfaz de depuración

Como se dijo anteriormente, Box2D no realiza ningún tipo de dibujo de los cuerpos físicos, sino que nosotros debemos encargarnos de ello. No obstante, para cuestiones de depuración, Box2D nos brinda algunas facilidades.

La clase `b2World` nos permite configurar una clase que implemente ciertos métodos requeridos que utilizará para dibujar los objetos físicos en pantalla. Es decir, debemos proveerle a `b2World` una clase con métodos para dibujar círculos, polígonos, etc. La implementación de esos métodos será nuestra tarea, porque sabemos cómo y dónde queremos dibujar, mientras que `Box2D` se encargará de llamar los métodos cuando desee dibujar alguna figura, sin saber cómo se encuentran implementados.

Para realizar esto, en primer lugar debemos crear la clase que se encargará de dibujar y servirá como nexo entre `Box2D` y `SFML`. Para ello, debemos generar una clase nueva que herede de la clase `b2DebugDraw` e implemente los métodos abstractos de `b2DebugDraw`. `b2DebugDraw` es una clase abstracta que tenemos que heredar e implementar. Imaginemos que creamos una clase:

```
class SFMLRenderer : public b2DebugDraw
{
private:
    RenderWindow *wnd;
public:
    SFMLRenderer(RenderWindow *window);
    ~SFMLRenderer(void);

    inline Color box2d2SFMLColor(const b2Color &_color);
    void DrawPolygon(const b2Vec2* vertices, int32 vertexCount, const
b2Color& color);
    void DrawSolidPolygon(const b2Vec2* vertices, int32 vertexCount, const
b2Color& color);
    void DrawCircle(const b2Vec2& center, float32 radius, const b2Color&
color);
    void DrawSolidCircle(const b2Vec2& center, float32 radius, const b2Vec2&
axis, const
b2Color& color);
    void DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color&
color);
    void DrawTransform(const b2Transform& xf);
    void DrawPoint(const b2Vec2& p, float32 size, const b2Color& color);
    void DrawString(int x, int y, const char* string, ...);
    void DrawAABB(b2AABB* aabb, const b2Color& color);
};
```

La misma implementa toda la funcionalidad que nos pide `Box2D` para dibujar sus primitivas. Esto sólo sirve para fines de depuración, ya que cuando estemos haciendo nuestros videojuegos, nosotros proveeremos los *sprites* correspondientes.

Dada esta clase, debemos decirle a `b2World` que tiene que usar sus métodos para dibujar los objetos de depuración. A esta acción la ejecutamos de la siguiente manera:

```
SFMLRenderer *debugRender;
phyWorld= new b2World(b2Vec2(0.0f,9.8f) ,true);
phyWorld->SetDebugDraw(debugRender);
```

Al hacer esto y correr el programa, veremos que se abre una ventana y que `Box2D` dibuja automáticamente los objetos que hemos creado. Pueden encontrar el código completo de esta clase en la plataforma del curso.

2.10. Box2d y SFML

Hasta aquí hemos visto cómo inicializar `box2d` y cómo trabajar con la interfaz de depuración y las clases que le hemos provisto. Pero, como dijimos anteriormente, en nuestro videojuego no utilizaremos esa interfaz, sino que nosotros seremos los responsables de unir el mundo de la visualización y el de la simulación.

Hasta este momento, cuando definíamos un objeto con representación gráfica en nuestro videojuego, declarábamos una clase que, además de la funcionalidad propia del objeto, poseía un *Sprite de SFML* que nos permitía dibujar el objeto. Imaginemos, por ejemplo, una clase que representa una pelota:

```
class Pelota{
private:
    Sprite _sprite;
    //otras variables
    //relacionadas al personaje
public:
    void Dibujar();
    //otros metodos
    //relacionados al personaje
};
```

De esta forma, encapsulábamos el dibujo de los sprites en cada objeto, es decir, uníamos el objeto con su representación gráfica.

Siguiendo esta idea, nuestra pelota, además de su representación gráfica, ahora puede incluir su objeto físico relacionado de box2d y encargarse de inicializarlo, configurarlo, actualizarlo y destruirlo. Además, deberá encargarse de actualizar la representación visual del objeto de física cuando éste cambie de posición.

Veamos, ahora, cómo podríamos implementar una clase Pelota que realice lo enunciado:

```

class Pelota{
private:
    //sprite y su imagen la para representación
    gráfica
    Sprite *_sprite;
    Image *_image;
    RenderWindow *_wnd;
    //body para representación física
    b2Body* _body;
    //...
public:
    Pelota(b2World* _world, RenderWindow
*_wnd){
        //guardamos puntero a ventana para
        dibujar luego
        wnd=_wnd;
        _image= newImage();
        _image->
>LoadFromFile("pelota.jpg");
        //cargamos el sprite
        _sprite= newSprite(*_image);

        //definimos el body y lo creamos
        b2BodyDefbodyDef;
        bodyDef.type = b2_dynamicBody;
        bodyDef.position.Set(100.0f,
0.0f);
        _body = _world->
>CreateBody(&bodyDef);

        //creamos su figura de colisión
        //en este caso suponemos que la
        figura de
        //colision es una caja cuadrada

        b2PolygonShape dynamicBox;
        dynamicBox.SetAsBox(20.0f, 20.0f);

        //creamos el fixture, le seteamos
        //la figura de colision
        //y agregamos el fixture al body
        b2FixtureDeffixtureDef;
        fixtureDef.shape = &dynamicBox;
        fixtureDef.density = 10.0f;
        fixtureDef.friction = 0.3f;
        fixtureDef.restitution=1.0f;
        _body->CreateFixture(&fixtureDef);

    }

    //metodo que posiciona el sprites
    //segun la posicion del body
    voidActualizarPosiciones(){
        b2Vec2pos=_body->GetPosition();
        _sprite->SetPosition(pos.x,pos.y);
    }

    //metodo que dibuja el sprite
    void Dibujar(){
        wnd->Draw(*_sprite);
    }
    //otros metodos
    //relacionados al personaje
};

```

Con esta clase veamos un ejemplo muy sencillo y simplificado sobre cómo llamaríamos los métodos para que todo funcione:

```

int _tmain(int argc, _TCHAR* argv[])
{
    //creamos la ventana de sfml
    RenderWindow *wnd=
newRenderWindow(VideoMode(800,600),"Simple Box
app");

    //creamos el world de box2d
    b2World* phyWorld=
newb2World(b2Vec2(0.0f,9.8f) ,true);

    //creamos una pelota indicandole la ventana
    y el world
    //para que pueda crear el sprite y el body

    Pelota* p1= newPelota(phyWorld,wnd);

    //loop principal
    while(wnd->IsOpened()){

        //avanzamos la simulación un
        tiempo igual al //tiempo de frame
        phyWorld->Step(wnd-
>GetFrameTime(),8,8);

        //ahora les decimos a los objetos
        que se //actualicen las posiciones de sus
        representaciones //graficas, ya que la simulación
        ya avanzó
        p1->ActualizarPosiciones();

        //y por ultimo le decimos que se
        dibujen
        wnd->Clear();
        p1->Dibujar();
        wnd->Display();

        //borramos las fuerzas para el
        próximo paso
        phyWorld->ClearForces();

    }
}

```

En el código se puede observar cómo, primero, se avanza la simulación; luego, de qué modo se actualizan las posiciones de los *sprites*, de acuerdo a la nueva posición de los *bodies*, y después, cómo se dibujan los objetos.

En este ejemplo no se realizó una orientación a objetos estricta, ya que implementamos todo en el main, simplemente para que resulte sencillo visualizarlo, pero esto *no* debe ser tomado como regla. Al ejecutar este ejemplo, deberían ver una pelota en caída libre que se va de pantalla, ya que no tiene con qué colisionar.

Estas ideas se pueden llevar al esqueleto de aplicación que utilizamos en la primera parte de la materia, con algunas modificaciones. En la plataforma se encuentra un proyecto que contiene un esqueleto de aplicación más adecuado para esta materia, sobre el cual pueden implementar esto que hemos estado desarrollando.

2.10.1. Fuerzas en SFML

Estudiaremos ahora cómo aplicar fuerzas simples, como las que hemos estudiado en esta unidad, utilizando box2d .

En box2d, las fuerzas y los momentos se aplican sobre el objeto *b2Body*, pero antes de ver cómo aplicar fuerzas, analicemos cómo configurar la masa de un cuerpo, ya que el movimiento dependerá de ella y de la fuerza.

En box2D tenemos dos formas de setear la masa. La forma natural de hacerlo es a través de los fixtures. Como vimos anteriormente, cada cuerpo posee varios fixtures y cada uno de ellos tiene una densidad determinada. La densidad de un cuerpo ρ y su masa m se relacionan de la siguiente manera:

$$\rho = \frac{m}{V}$$

Dónde V es el volumen del cuerpo. Es decir, la densidad es una magnitud escalar que expresa la relación entre la masa y el volumen de un cuerpo. Su unidad en el Sistema

Internacional es el kilogramo por metro cúbico $\frac{kg}{m^3}$. En nuestro caso, como estamos en un mundo en 2d, el volumen de un cuerpo es directamente su área, por lo que a box2d

le pasaremos la densidad en las unidades $\frac{kg}{m^2}$.

Utilizando la densidad de los fixtures y las geometrías de colisión, Box2d determina la masa de cada fixture y luego las suma para obtener la masa total del cuerpo. Esta es la forma automática de calcularla.

Por otro lado, podemos setear manualmente la masa mediante la función:

```
voidSetMassData(const b2MassData* data);
```

Pero para ello, no sólo tenemos que pasarle la masa, sino también el tensor de inercia, que veremos en las próximas unidades. Así que, por el momento, dejaremos que box2d calcule la masa automáticamente.

Existen varias funciones para agregarle fuerzas a un b2Body, pero por el momento utilizaremos la siguiente:

```
voidApplyForce(const b2Vec2& force, const b2Vec2& point);
```

La misma recibe como primer argumento un vector de fuerza y, como segundo argumento, un punto que indica dónde se aplica la fuerza. Por ahora, ignoraremos el segundo argumento y más adelante veremos para qué sirve. Mientras tanto, utilizaremos siempre el vector O.

BIBLIOGRAFÍA

Gettys, W.E.; Sélér, F.J.; Skove, M.J. *Física Clásica y Moderna*. Madrid, McGraw-Hill Inc., 1991.

Sears, F.; Zemansky, M.; Young, H.; Freedman, R. *Física Universitaria*. Vol. 1, Addison Wesley Longman, 1998.

Resnic; Halliday. *Física para estudiantes de Ciencias e Ingeniería*. Parte I. México, Compañía Editorial Continental S.A., 1967.

Alonso, Finn. *Física: Vol. I: Mecánica*. Fondo Educativo Interamericano, 1970.

Botto, J.; González, N.; Muñoz, J. *Fís Física*. Buenos Aires, Tinta Fresca, 2006.

Gaisman, M.; Waldegg Casanova, G.; Adúriz-Bravo, A.; Díaz, F.; Lerner, A.; Rossi, D. *Física. Movimiento, interacciones y transformaciones de la energía*, Buenos Aires, Santillana, 2007.