UNIVERSIDAD CATÓLICA DE TEMUCO
INGENIERÍA CIVIL EN INFORMÁTICA
FACULTAD DE INGENIERÍA

# Red Bounce Ball

Luciano Ignacio Revillod Jerez

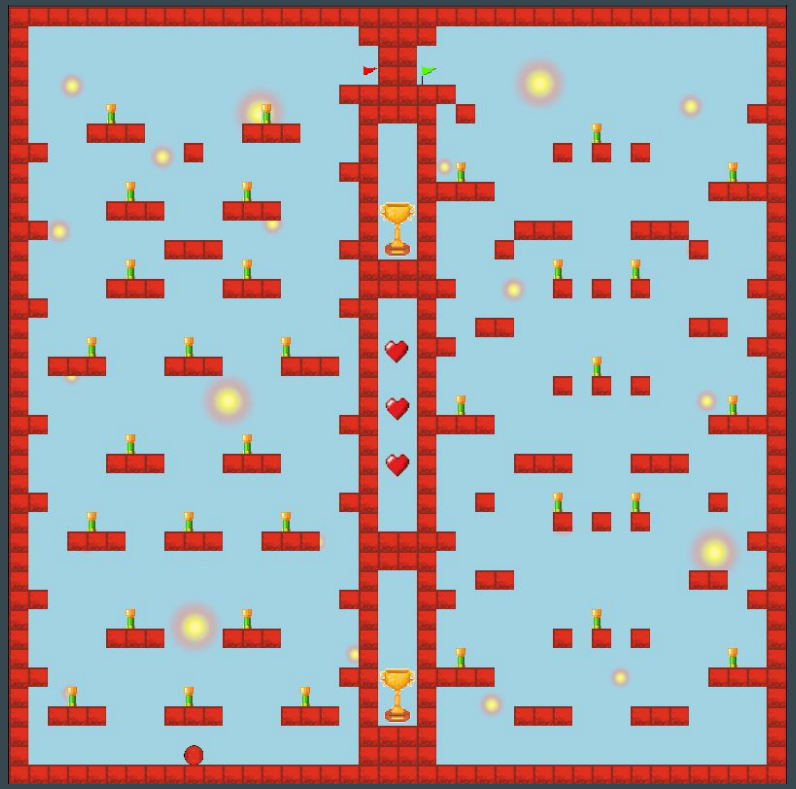
Programación I, Sección III, 2022

# Introducción

Red Bounce Ball es un videojuego basado en pygame que está compuesto por múltiples clases, funciones y ciclos.

El objetivo principal del proyecto es demostrar que con conocimientos básicos sobre el lenguaje de programación python se pueden obtener grandes resultados.

La idea del proyecto surge en base a los contenidos vistos en la asignatura de Programación I y Programación de Robot, además de homenajear al juego original de Nokia, Bounce Tales.



# Clases en python

En Python, las clases son un método utilizado para trabajar con objetos. Al crear una clase, se crea un nuevo tipo de objeto, permitiendo crear nuevas instancias de ese tipo. Cada instancia de clase puede tener atributos adjuntos para mantener su estado. Las instancias de clase también pueden tener métodos (definidos por su clase) para modificar su estado.

El término self corresponde a un prefijo asignado a los parámetros de la clase, se debe asignar como parámetro de entrada y acompañar a los parámetros correspondientes.

## Inicialización de parámetros en una clase:

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
```

## Actualización de parámetros en una clase:

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

# Clases en Red Bounce Ball

El proyecto Red Bounce Ball se compone de dos clases principales:

- Clase World: Esta clase contiene los parámetros y funciones necesarias para la creación del mapa.

- Clase Redball: Esta es una clase multipropósito, pero principalmente se encarga de la actualización de posición del jugador así como de las transiciones entre niveles y creación de botones.

```
class world():
    def __init__(self,matriz,sprt):
        self.listaTile = [] #..... Lista que contiene las plataformas/bordes del mapa.
        self.deadpoint = [] #..... Lista que contiene los objetos "deadpoint".
        self.flag = []      #..... Lista que contiene la bandera roja.
        self.final = []     #..... Lista que contiene la bandera verde.
        filas = 0           #..... Contador de la cantidad de filas de la matriz.
        for fila in matriz:
            columnas = 0    #..... Contador de la cantidad de columnas de la matriz
            for i in fila:  #..... Se recorre uno a uno los subindices de la matriz
                if i == 1:  # (4)
                    img = pygame.transform.scale(sprt[1], (20, 20)) # (5)
                    img_rect = img.get_rect()                       # (6)
                    img_rect.x = columnas * 20                      # (7)
                    img_rect.y = filas * 20                         # (8)
                    tile = (img, img_rect)                          # (9)
                    self.listaTile.append(tile)
                if i == 2:                                          # (4)
                    img = pygame.transform.scale(sprt[2], (20,20))  # (5)
                    img_rect = img.get_rect()                       # (6)
                    img_rect.x = columnas * 20                      # (7)
                    img_rect.y = filas * 20                         # (8)
                    tile = (img, img_rect)                          # (9)
                    self.flag.append(tile)
                if i == 3:                                          # (4)
                    img = pygame.transform.scale(sprt[3], (10,20))  # (5)
                    img_rect = img.get_rect()                       # (6)
                    img_rect.x = columnas * 20                      # (7)
                    img_rect.y = filas * 20                         # (8)
                    tile = (img, img_rect)                          # (9)
                    self.deadpoint.append(tile)
                if i == 4:                                          # (4)
                    img = pygame.transform.scale(sprt[9], (20,20))  # (5)
                    img_rect = img.get_rect()                       # (6)
                    img_rect.x = columnas * 20                      # (7)
                    img_rect.y = filas * 20                         # (8)
                    tile = (img, img_rect)                          # (9)
                    self.final.append(tile)
                columnas += 1
            filas += 1
        return
```

# Matriz de la clase world

La finalidad de esta matriz es poder posicionar objetos estáticos de forma sencilla en el mapa. Esta matriz tiene un tamaño de 40x40.

```
matriz = [
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,1,4,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,3,0,0,0,0,0,0,0,3,0,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,0,0,1,1],
    [1,0,0,0,1,1,1,0,0,0,0,0,1,1,1,0,0,0,1,0,0,1,0,1,0,0,0,0,0,0,3,0,0,1],
    [1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,0,1,0,1,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,3,0,0,0,0,0,0,0,0,3,1],
    [1,0,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,3,0,0,0,1],
    [1,0,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,1,1,0,0,0,1,1,1],
    [1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,1],
    [1,0,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,3,0,0,0,1],
    [1,0,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,1,0,0,0,0,0,0,0,1,1],
    [1,0,0,0,3,0,0,0,0,3,0,0,0,0,3,0,0,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0,3,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,1,0,1,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,3,0,0,0,0,0,0,0,0,0,3,1],
    [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,1,1],
    [1,0,0,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,1,0,0,0,0,1,1,1,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,3,0,0,0,3,0,0,1,1],
    [1,0,0,0,3,0,0,0,0,3,0,0,0,0,3,0,0,0,1,0,0,1,0,0,0,0,0,1,0,1,0,1,0,1],
    [1,0,0,1,1,1,0,0,1,1,1,0,0,0,1,1,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,1,0,0,0,0,0,0,0,1,1],
    [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1],
    [1,0,0,0,0,0,3,0,0,0,0,0,3,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,1,0,1,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,3,0,0,0,0,0,0,0,0,0,3,1],
    [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,1,1],
    [1,0,0,3,0,0,0,0,3,0,0,0,0,3,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0,1,1,1,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
    ]
```

# Clase Redball

Función __init__:

En la clase Redball, la función __init__ corresponde a la base de los parámetros de nuestro objeto.

La clase recibe los datos de entrada self, x, y, donde x e y corresponden a la posición inicial en los respectivos ejes del objeto en el mapa.

Se carga una imagen, se reescala y mediante la función “get_rect” se obtiene la hitbox/ contorno del sprite.

Self.salto es una forma de comprobar si el jugador ha saltado, inicialmente es Falso.

Self.vida corresponde a la cantidad de vidas que posee el jugador(3).

```
class Redball():
    def __init__(self, x, y): #.........................
        img = pygame.image.load("./sprt/S07.png") #.....
        self.rescale = pygame.transform.scale(img, (20, 20))
        self.rect = self.rescale.get_rect() #...........
        self.ancho = self.rescale.get_width() #.........
        self.alto = self.rescale.get_height() #.........
        self.rect.x = x #...............................
        self.rect.y = y #...............................
        self.gravity = 0 #..............................
        self.salto = False #............................
        self.vida = 3  #................................
```

# Clase Redball

Función update:

En la clase, la función update corresponde a la actualización de la posición del jugador, también se encarga de cambiar los estados e interacción mediante botones con el usuario.

Para la relación usuario-programa se utiliza la funcion pg.key.get_pressed()

Esta función detecta la pulsación de teclas del teclado, si a esta función le sumamos un condicional y una consecuencia, obtenemos una actualización en el programa en base a algo tangible.

```
#=============================================================================================#
#                                        Funcion update                                       #
#=============================================================================================#

    def update(self,Screen,mundo,sprt):
        dirX = 0 #..............(movimiento)............... Direccion en X inicial.
        dirY = 0 #..............(movimiento)............... Direccion en Y inicial.

#=============================================================================================#
#                              Se verifica la pulsacion de teclas.                            #
#=============================================================================================#

        if self.vida == 3 or self.vida == 2 or self.vida == 1: #.... Check de las vidas del jugador.
            cupblit(Screen,sprt) #................................. Blit de las copas.
            vidasblit(self,Screen,sprt)#........................... Blit de las vidas.

            key = pygame.key.get_pressed() #....................... Verificador de teclas presionadas.
            if key[pygame.K_UP] and self.salto == False: #......... Tecla Arriba
                self.gravity = -13 #.....  Se resta 13 a la "gravedad", de esta forma la pelota subira
                self.salto = True #.................................. Check de salto
            elif key[pygame.K_LEFT]:
                dirX -= 5 #.......................................... Direccion a la izquierda
            elif key[pygame.K_RIGHT]:
                dirX += 5 #.......................................... Direccion a la derecha
```

# Función update

Actualización de gravedad:

En este apartado es cuando el nombre del parámetro self.gravedad toma sentido.

Digamos que saltamos, nuestro objeto subirá, pero hasta ahora no hay nada que lo haga bajar, por eso si constantemente se le suma una cierta cantidad a la gravedad, una vez que el objeto haya alcanzado su altura máxima este tenderá a bajar.

Colisiones:

Para detectar colisiones existen múltiples métodos, en este caso se utilizó collide rect, a esta función se le asignan dos objetos con rect de la forma “rect1.collide rect(rect2)”

```
#=============================================================================================#
#                                  Actualizacion de "Gravedad"                                #
#=============================================================================================#

self.gravity += 1 #... En todo momento +1 de esta forma, aunque la pelota suba, va a bajar.
if self.gravity > 10: #... Si la suma constante es mayor a
    self.gravity = 10 #... Solo un limitador
dirY += self.gravity  #... La gravedad va a ser igual a nuestra direccion en Y (Movimiento)

#=============================================================================================#
#                                        DEADPOINTS                                           #
#=============================================================================================#

for i in mundo.deadpoint: #..... Se recorren los elementos de la lista deadpoint.
    if i[1].colliderect(self.rect.x + dirX, self.rect.y, self.ancho, self.alto):
        self.vida -= 1 #............... Se resta una vida al contador.
        dirX = 0 #..................... Frena el movimiento en eje X
        dirY = 0 #..................... Frena el movimiento en eje Y
        self.gravity = 0  #............ Frena la "Gravedad"
        self.rect.x = self.rect.x #.... Retorna posicion de inicio en X
        self.rect.y = 775 #............ Retorna posicion de inicio en Y

    if i[1].colliderect(self.rect.x, self.rect.y + dirY, self.ancho, self.alto):
        self.vida -= 1 #............... Se resta una vida al contador.
        dirX = 0 #..................... Frena el movimiento en eje X
        dirY = 0 #..................... Frena el movimiento en eje Y
        self.gravity = 0  #............ Frena la "Gravedad"
        self.rect.x = self.rect.x #.... Retorna posicion de inicio en X
        self.rect.y = 775 #............ Retorna posicion de inicio en Y

#..............Actualizacion de las coordenadas de la pelota.........#

self.rect.x += dirX
self.rect.y += dirY
Screen.blit(self.rescale, self.rect) #.... Blit del sprt sobre la posicion de la pelota.

if self.rect.y < 0 or self.rect.y > 780: #.... Condicional creado para solucionar un error.
    self.rect.y = 775
    self.rect.x = self.rect.x
```

# Función update

Colisión con la meta:

Para esta sección del código, además de la función colliderect existe la inclusión de botones, de manera que cada vez que nuestro player colisione con la meta se hará un blit sobre la pantalla el cual incluirá dos botones, uno que mediante pg.display.quit permitirá cerrar el programa y otro que reiniciara los parámetros de la pelota, haciendo que se retorne al nivel uno.

```
#=============================================================================#
#                          Banderas de END GAME                               #
#=============================================================================#

for i in mundo.final:
    #check de colision en eje X e Y
    if i[1].colliderect(self.rect.x + dirX, self.rect.y, self.ancho, self.alto): # Eje X
        pygame.mouse.set_visible(True) #......... Mouse se vuelve visible
        b_vamos = botones(430,650, sprt[8]) #.... Defino los botones con su respectiva posicion
        b_salir = botones(150,650, sprt[22])#.... (Volver a jugar y salir)
        Screen.blit(sprt[10],(0,0)) #............ Blit del background final (Ganador!)
        if b_vamos.draw(Screen):    #............ Si se presiona el boton:
            pygame.mouse.set_visible(False)
            self.rect.x = 180
            self.rect.y = 760
            self.gravity = 0        #...Se reinician los parametros de la pelota...#
            self.salto = False
            self.vida = 3
        if b_salir.draw(Screen):    #............ Si se presiona el boton:
            pygame.display.quit()   #............ Se cierra el juego.

for i in mundo.final:
    #check de colision en eje X e Y
    if i[1].colliderect(self.rect.x, self.rect.y + dirY, self.ancho, self.alto): # Eje Y
        pygame.mouse.set_visible(True) #......... Mouse se vuelve visible
        b_vamos = botones(430,650, sprt[8]) #.... Defino los botones con su respectiva posicion
        b_salir = botones(150,650, sprt[22])#.... (Volver a jugar y salir)
        Screen.blit(sprt[10],(0,0)) #............ Blit del background final (Ganador!)
        if b_vamos.draw(Screen):    #............ Si se presiona el boton:
            pygame.mouse.set_visible(False)
            self.rect.x = 180
            self.rect.y = 760
            self.gravity = 0        #...Se reinician los parametros de la pelota...#
            self.salto = False
            self.vida = 3
        if b_salir.draw(Screen):    #............ Si se presiona el boton:
            pygame.display.quit()   #............ Se cierra el juego.
```

# Función update

Vidas agotadas:

Si recordamos el inicio de la clase update habia un condicional que reconocía los casos donde el jugador tenía hasta tres vidas, la contraparte de este condicional es donde el jugador se queda sin vidas, en este caso ocurre algo similar a las secciones anteriores en donde se hace un blit que indica que el jugador se queda vidas además de dos botones que permiten cerrar el juego o bien volver volver a intentar.

Es importante recordar que la función update está dentro del bucle principal del código, por lo tanto estas comprobaciones se estarán realizando en todo momento.

```
#=============================================================================
                                Cero vidas
#=============================================================================

elif self.vida == 0 or self.vida < 0: #.............. Si se agotan las vidas:
    pygame.mouse.set_visible(True) #.................. Mouse se vuelve visible.
    b_reintentar = botones(430,650, sprt[20])#....... Defino los botones con su
    b_salir = botones(150,650, sprt[22]) #........... (Volver a jugar y salir)
    Screen.blit(sprt[19],(0,0))#...................... Blit del background final
    if b_reintentar.draw(Screen): #................... Si se presiona el boton:
        pygame.mouse.set_visible(False)
        self.rect.x = 180
        self.rect.y = 760
        self.gravity = 0          #...Se reinician los parametros de la pelota..
        self.salto = False
        self.vida = 3

    if b_salir.draw(Screen):      #............ Si se presiona el boton:
        pygame.display.quit()     #............ Se cierra el juego.
```

# Conclusión

Si bien pygame no es la forma más óptima de desarrollar un videojuego esta librería nos permite entender el cómo funcionan diferentes aspectos de un juego básico como lo es Red Bounce Ball.

La elección de este proyecto contrajo importantes desafíos, solución de errores y aprendizajes externos a la programación como lo son el uso de software de edición de imágenes.