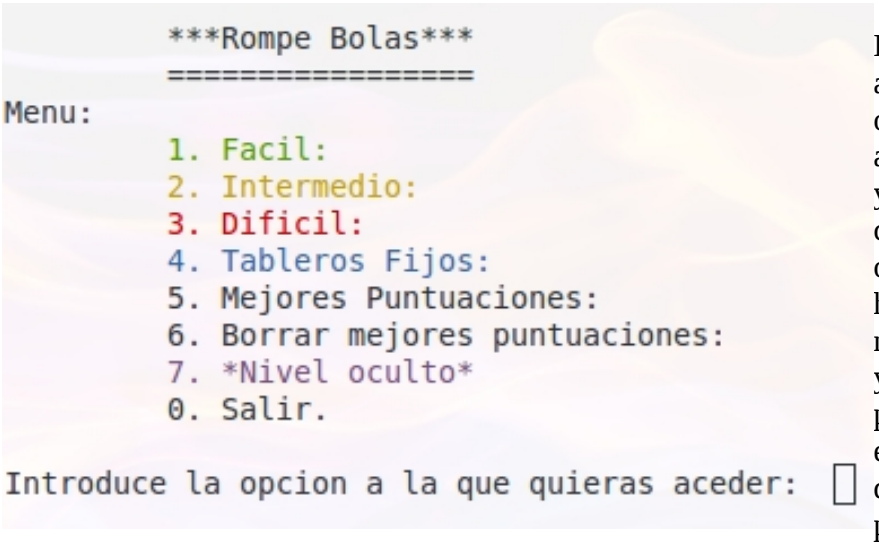
Practica 1
Paradigmas de Programación
Rompe Bolas

Jose Luis Martín Martín y Daniel Jimenez Cubo.

Simulación de juego:

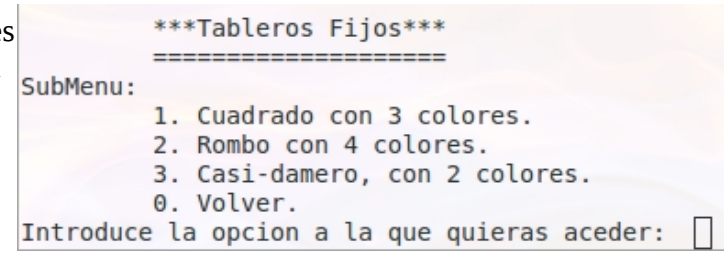
Nada mas iniciar el programa se cargan los datos de las puntuaciones previas en otras partidas, solo se da algún tipo de información por pantalla al usuario si no se puede acceder por algún motivo al archivo que guarda las puntuaciones, en ese caso se crea un archivo nuevo en el directorio donde ejecutamos el programa para guardas las puntuaciones actuales. Después de esto se muestra el menú principal del programa el cual tiene la siguiente apariencia:



En la imagen podemos ver que además de las opciones obligatorias hay una opción 7 añadida que solo se muestra tal y como esta en la imagen cuando se hayan jugado a los otros 6 niveles anteriormente, hasta ese momento la opción 7 ni siquiera aparecerá en el menú y no sera accesible. Al borrar las puntuaciones también se elimina el acceso al nivel oculto hasta que se realicen de nuevo partidas completas de los 6 niveles correspondientes.

Si seleccionamos la opción 4 la apariencia del sub menú de los tableros fijos sera la siguiente:

Contamos con tres tableros fijos los cuales tiene siempre el mismo diseño (aunque la opción 3 del casi-damero tiene una componente aleatoria como se indica en las especificaciones), la opción 0 nos retornaría al menú principal visto anteriormente.



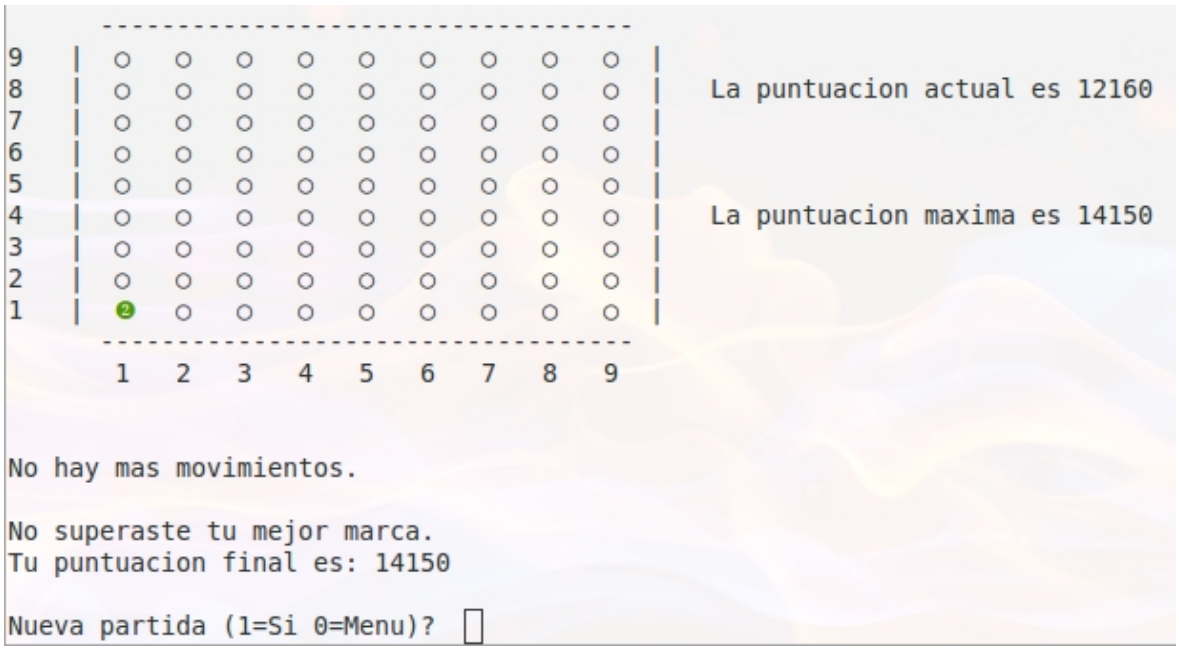
Como podemos ver en la anterior imagen esa es la apariencia típica que tienen nuestros tableros, para representar las bolas utilizamos caracteres con formas redondas y un numero identificativo del

tipo de bola, además de un color asociado al tipo de bola que representa. Las puntuaciones actuales y máximas de este modo quedan a la derecha del tablero y en la parte inferior de la pantalla no aparecerá un mensaje para introducir las coordenadas del siguiente movimiento (Hasta que no haya posibilidad de mas movimientos.), las coordenadas se introducen ambas en la misma linea de edición con ambos números separados por un espacio como se indicara por pantalla, en caso de que el movimiento no genere ningún cambio en el tablero se volverá a mostrar de nuevo para introducir la siguiente jugada. También reconocerá los valores no permitido volviendo a pedir una nueve entrada de datos.

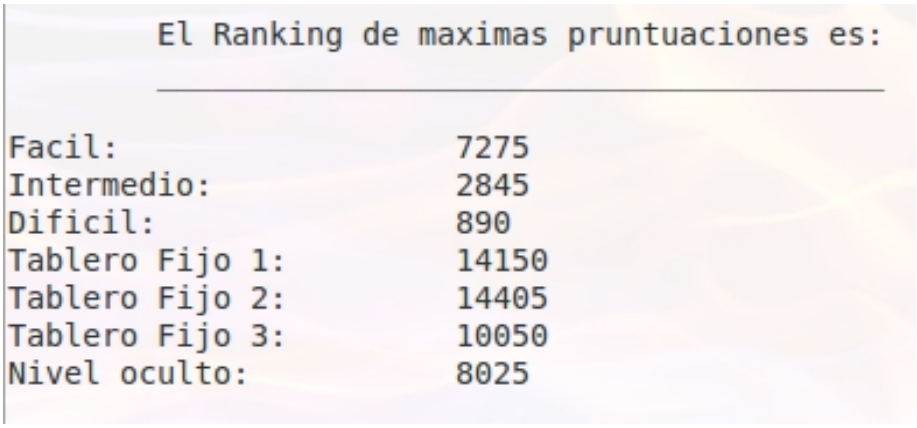
Tras realizar la jugada 2 2 eliminamos las bolas verdes y nos quedaría como vemos en la imagen de arriba.

En la siguiente jugada eliminamos las bolas amarillas con las coordenadas 2 3 y nos queda el tablero en la situación de la imagen superior.

Por ultimo en esta partida solo queda eliminar el gran bloque que forman las bolas rojas, seleccionamos en cualquier punto de ese bloque y acabaríamos la partida, nos mostraría el tablero ya sin movimientos, la puntuación final que hemos obtenido, y por ultimo nos preguntaría si queremos realizar otra partida en dicho modo o volver al menú (En cualquier modo esta preestablecido que se redirija al menú principal, aunque sea un tablero fijo).



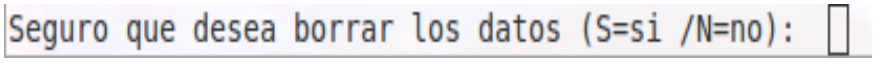
A continuación mostraremos la apariencia que tiene los modos de mostrar puntuaciones y borrar puntuaciones.



Como podemos ver al seleccionar la opción 5 del menú nos muestra una tabla con las puntuaciones obtenidas en cada modalidad, sin necesidad de realizar mas pasos nos las muestra y nos manda de nuevo al menú ya que en este paso no se requiere ninguna interacción mas con el usuario.

Por ultimo la opción 6 del menú la cual se emplea para borrar las puntuaciones, en nuestro programa tiene la mejora de realizar un paso intermedio al borrar las puntuaciones con un mensaje de confirmación antes de borrar las puntuaciones, si seleccionamos el no volvernos al menú y si seleccionamos en su lugar el si simplemente nos mostrara un mensaje además informándonos de que las puntuaciones se han borrado y nos redirigirá al menú.

A continuación podemos ver el mensaje que nos muestra para pedir la confirmación de borrar puntuaciones.



Pruebas:

Selección de opción en el menú principal:

Caso:	Entrada:	Salida Esperada:	Salida Obtenida:
1	1	Modo correspondiente: fácil.	Modo correspondiente: fácil.
2	2.5	Introducir de nuevo (Valor no valido)	Introducir de nuevo (Valor no valido)
3	1 4	Introducir de nuevo (Valor no valido)	Introducir de nuevo (Valor no valido)
4	14	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
5	7 (Deshabilitada.)	Introducir de nuevo (Valor no valido)	Introducir de nuevo (Valor no valido)
6	7 (Habilitada.)	Modo correspondiente: nivel oculto.	Modo correspondiente: nivel oculto.
7	siete	Introducir de nuevo (Valor no valido)	Introducir de nuevo (Valor no valido)
8	5	Modo correspondiente: mostrar puntos.	Modo correspondiente: mostrar puntos.
9	6	Modo correspondiente: borrar puntos.	Modo correspondiente: borrar puntos.
10	0	Salir. Terminar ejecución.	Salir. Terminar ejecución.
11	-1	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
12	“\n”	Introducir de nuevo (Valor no valido)	Introducir de nuevo (Valor no valido)

Selección de opción en el sub menú:

Caso:	Entrada:	Salida Esperada:	Salida Obtenida:
1	1	Modo correspondiente: tab fijo 1.	Modo correspondiente: tab fijo 1.
2	2.5	Introducir de nuevo (Valor no valido)	Introducir de nuevo (Valor no valido)
3	-1	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
4	14	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
5	cuatro	Introducir de nuevo (Valor no valido)	Introducir de nuevo (Valor no valido)
6	0	Salir sub menú volver al menú.	Salir sub menú volver al menú.

Introducir jugadas en los distintos modos:

Caso:	Entrada:	Salida Esperada:	Salida Obtenida:
1	1,0	Introducir de nuevo (formato no valido)	Introducir de nuevo (formato no valido)
2	1 0	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
3	10	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
4	1 1	Valores validos (jugada 1 1)	Valores validos (jugada 1 1)
5	10 10	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
6	'a' 'b'	Introducir de nuevo (formato no valido)	Introducir de nuevo (formato no valido)
7	0, 0	Introducir de nuevo (formato no valido)	Introducir de nuevo (formato no valido)
8	0 0	Salir directamente al menú principal.	Salir directamente al menú principal.
9	-3 5	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
10	-34	Introducir de nuevo (Fuera de rango)	Introducir de nuevo (Fuera de rango)
11	salir	Introducir de nuevo (formato no valido)	Introducir de nuevo (formato no valido)
12	00	Introducir de nuevo (formato no valido)	Introducir de nuevo (formato no valido)

Estructura de datos empleada:

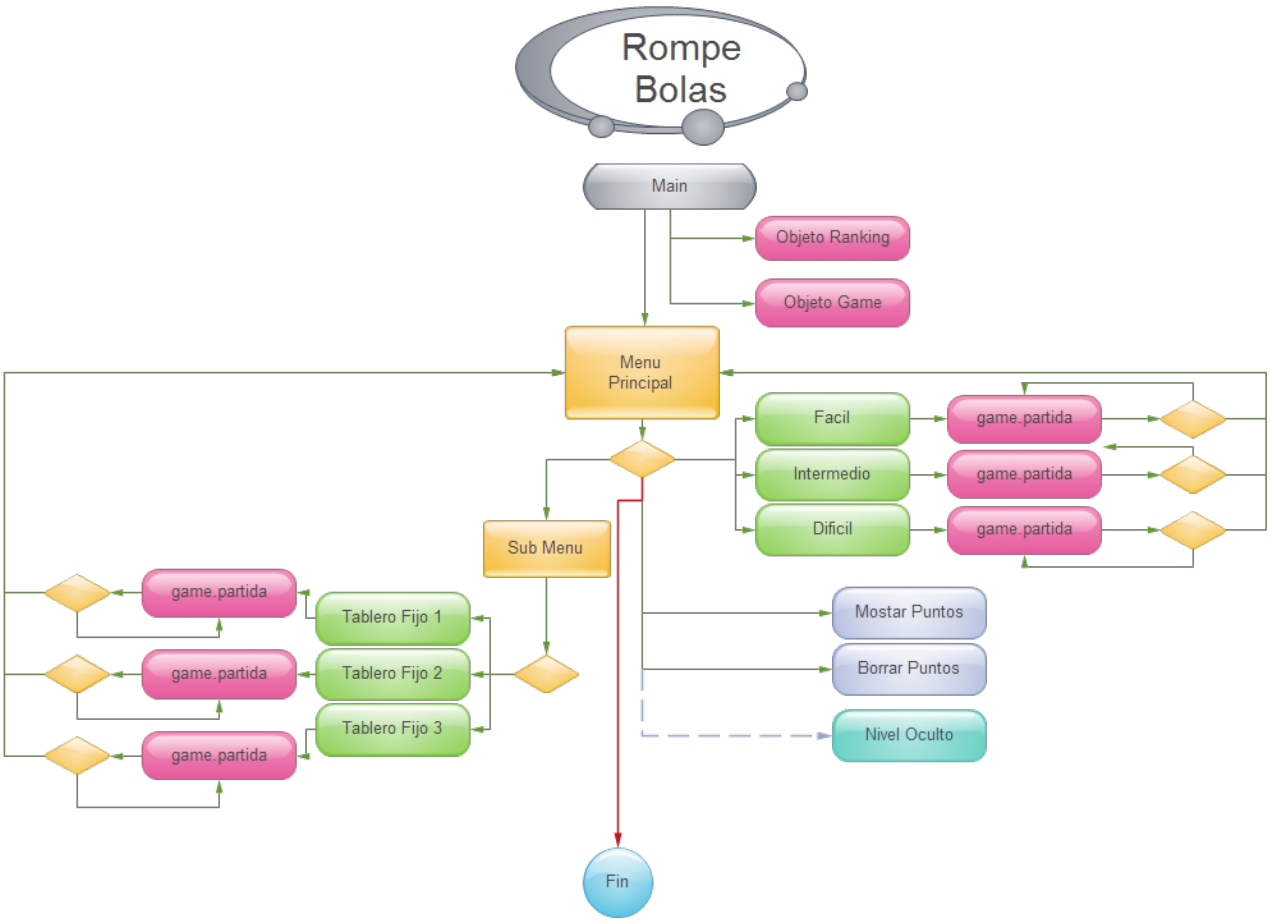
El programa consta de tres archivos python el main.py es la parte principal del programa y para su ejecución utiliza un objeto de la clase Game que viene definida en el archivo game.py, también emplea un objeto de la clase Ranking que viene definida en el archivo rank.py.

El archivo main cuenta con las dos funciones principales que son el menú principal que nos ofrece la posibilidad de seleccionar los distintos modos de juego, abrir el sub menú del programa, acceder o borrar las mejores puntuaciones y poder acceder al nivel oculto del juego.

Las funciones del objeto Game son las encargadas de simular las partidas del juego, cada objeto Game cuenta con una matriz 9x9 que hace de tablero de juego y dos variables de instancia que se emplean como contadores del numero de 0s en cada momento y los puntos de la partida.

El objeto Ranking y las funciones que utilizan se emplean para leer y escribir datos del archivo "puntuaciones.txt", contiene los resultados de los seis modos de juegos habituales, mas la puntuación del nivel extra y las funciones encargadas de mostrar y borrar las puntuaciones que se llaman desde el menu principal.

Módulos:



Archivo main:

Parte principal del programa consta de dos métodos uno por cada menú del juego, y la llamada desde el main al menú principal que es el que inicia la partida. Además importa las clases necesaria para el funcionamiento del programa.

```
from game import Game
from rank import Ranking
```

Y definimos un objeto de cada clase que son los que usaremos para jugar y movernos por las distintas funciones.

```
mimatriz= Game()
mispuntos= Ranking()

menuprincipal()     #Llamada al menú para iniciar el juego.
```

Método ***menuprincipal***:

bucle:

print "..." *#Mostramos las opciones del menú*

menu1=**raw_input**("Selecione la opción elegida:")

try:

Comprobamos que sea un valor correcto.

except:

Si no ha sido un valor correcto o dentro del rango no salimos del bucle.

#según el dato introducido entra en la opción que le corresponda.

if(menu1 == 1): *#Entra en la opción 1*

bucle:

jugar una partida de este tipo mientras el usuario no quiera salir al final.

elif(menu1 == 2): *#Entra en la opción 2*

…

…

esle: *#Entra en la ultima opción*

…

Método ***submenu***:

bucle:

print "..." *#Mostramos las opciones del submenú*

menu1=**raw_input**("Selecione la opción elegida:")

try:

Comprobamos que sea un valor correcto.

except:

Si no ha sido un valor correcto o dentro del rango no salimos del bucle.

#según el dato introducido entra en la opción que le corresponda.
if(menu == 1): *#Entra en la opción 1*
    bucle:
        jugar una partida con tablero fijo 1 mientras el usuario no quiera salir al final.

elif(menu == 2): *#Entra en la opción 2*
    bucle:
        jugar una partida con tablero fijo 2 mientras el usuario no quiera salir al final.

elif(menu == 3): *#Entra en la opción 3*
    bucle:
        jugar una partida con tablero fijo 3 mientras el usuario no quiera salir al final.

else: *#Sale de este submenu y vuelve al menuprincipal*
    menuprincipal()

Archivo game:

Archivo game.py contiene a la clase Game del juego.

- La cual cuenta con el tablero de juego actual formado por una lista bidimensional de enteros, inicialmente con todos sus valores puestos a 0.
- También tiene otras dos variables de instancia las cuales en cada nueva partida que se inicia se restablecen a 0 las dos y cuentan la puntuación actual y el numero de 0s (Huecos en el tablero) que tiene la partida en cada momento de juego.

```
import random
```

Se importa la clase **random** necesaria para alguno de los métodos dentro de la clase Game.

Método **inicializador(__init__)**:

```
self.mat = [ [ 0 for x in range(9)] for x in range(9)]   #Iniciamos la matriz todo a 0
self.nceros = 0                                          #Iniciamos también los puntos y
self.puntos = 0                                          #numero de 0 también a 0
```

Método **facil**:

```
self.nceros = 0                                          #Ponemos estas variables de
self.puntos = 0                                          #nuevo a 0 por si venimos de una
                                                         #partida anterior.
for con i recorremos las filas de self.mat:
    for con j recorremos las columnas de self.mat:
        self.mat [i][j] = random.randrange(1,4)          #Damos a cada puesto de la
                                                         #matriz un valor aleatorio entre
                                                         #1 y 3
```

Método **intermedio**:

```
self.nceros = 0                                  #Ponemos estas variables de
self.puntos = 0                                  #nuevo a 0 por si venimos de una
                                                 #partida anterior.
for con i recorremos las filas de self.mat:
        for con j recorremos las columnas de self.mat:
                self.mat [i][j] = random.randrange(1,5)   #Damos a cada puesto de la
                                                          #matriz un valor aleatorio entre
                                                          #1 y 4
```

Método **dificil**:

```
self.nceros = 0                                  #Ponemos estas variables de
self.puntos = 0                                  #nuevo a 0 por si venimos de una
                                                 #partida anterior.
for con i recorremos las filas de self.mat:
        for con j recorremos las columnas de self.mat:
                self.mat [i][j] = random.randrange(1,6)   #Damos a cada puesto de la
                                                          #matriz un valor aleatorio entre
                                                          #1 y 5
```

Método **tf1**:

```
self.nceros = 0                                  #Ponemos estas variables de
self.puntos = 0                                  #nuevo a 0 por si venimos de una
                                                 #partida anterior.
x=0
while(x<5):
        for i en el rango (x, numero de filas -x):        #realizamos cada ciclo del bucle
                for j en el rango (x, numero de columnas -x):   #por cada cuadrado del tablero
                        if(x==0) self.mat[i][j] =1              #de un color y vamos reduciendo
                        elif(x==1) self.mat[i][j]=2             #los valores de la matriz que
                        elif(x==2) self.mat[i][j]=3             #recorremos en cada vuelta.
                        elif(x==3) self.mat[i][j]=1
                        else self.mat[i][j]=2
        x += 1
```

Método **tf2**:

```
self.nceros = 0                                  #Ponemos estas variables de
self.puntos = 0                                  #nuevo a 0 por si venimos de una
                                                 #partida anterior.

for con i recorremos las filas de self.mat:      #Llenamos la matriz de 4 la bola
        for con j recorremos las columnas de self.mat:   #mas abundante y el resto las
                self.mat [i][j] = 4                      #las introducimos a mano.
self.mat[0][4]=1
…
```

Método **tf3**:

```
self.nceros = 0                                  #Ponemos estas variables de
self.puntos = 0                                  #nuevo a 0 por si venimos de una
                                                 #partida anterior.

for con i recorremos las filas de self.mat:      #Recorremos la matriz, dejando
        for con j recorremos las columnas de self.mat:   #pares con 1 e impares con 2,
                if(((i+j)%2)==0) self.mat [i][j] =1      #como un damero.
                else self.mat[i][j] = 2
i=random(entre 0 y 9)                            #Cambiamos el valor de esa bola
j=random(entre 0 y 9)                            #Para poder tener movimientos.
if(self.mat[i][j]==1):   self.mat[i][j]=2
else:  self.mat[i][j]=1
```

Método **niveloculto**:

```
self.nceros = 0                                  #Ponemos estas variables de
self.puntos = 0                                  #nuevo a 0 por si venimos de una
                                                 #partida anterior.

for con i recorremos las filas de self.mat:      #Llenamos por defecto con 1s
        for con j recorremos las columnas de self.mat:   #las bolas de las diagonales
                self.mat[i][j]=1                 #con 4s
                if ((float(i+j)/2=i) | (float(i+j)/2=4)) self.mat[i][j] = 4
                if(i==0 | i==8) self.mat[i][j] = 3       #Llenamos los bordes con 2s y 3s
                if(j==0 | j==8) self.mat[i][j] = 2
self.mat[0][8]==3
```

Método **mostrar**:

#según el modo de juego nos da la posición de la mejor puntuación histórica en ese modo.

```
for con i recorremos las filas de self.mat:      #recorre la matriz y va
        for con j recorremos las columnas de self.mat:   #mostrando su contenido
                        print self.mat[i][j]
                        if(i==7) print(Puntuación actual)    #Muestra las puntuaciones a un
                        if(i==3) print(Puntuación máxima)    #lado del tablero.
        print “”                                 #Al terminar fila imprime un
                                                 #salto de linea
```

Método **cerosup**:

```
cer = [0 for range(len(mat))]            #Creamos una lista de 0s con el tamaño igual al
                                         #numero de filas de la matriz.

for con i recorremos las filas de self.mat:      #Recorremos la matriz contando
        for con j recorremos las columnas de self.mat:   #el numero de ceros por columna
                cer[j] = cer[j]+1                #y lo almacenamos en la lista.
for realizamos esta tarea tantas veces como filas tiene para evitar errores.
        for con i recorremos las filas de self.mat:
```

```
            for con j recorremos las columnas de self.mat:
                if(self.mat[i][j]==0):
                    for k desde i hasta la penúltima posición:
                        self.mat[k][j]=self.mat[k+1][j]
```

#recorremos la matriz cada vez que encontramos un 0 bajamos los valores que hay por encima de el #hasta dejar la matriz con 0s únicamente en la fila mas alta.

```
for con j recorremos las columnas de self.mat:          #recorremos la matriz de nuevo llenado
    for con i recorremos las filas en orden inverso:    #la matriz de tantos ceros como tenia.
        if(cer[j]!=0):
            self.mat[i][j]=0
            cer[j] = cer[j] -1
```

Metodo **columns**:

```
c = 0   #Empezamos creando una variable para contar el numero de columnas con todos
        #sus elementos 0s
for con i recorremos las filas de self.mat:                 #Recorremos la matriz, si
    for con j recorremos las columnas de self.mat:          #una columna tiene todos sus
        ...                                                 #elementos 0 c +=1.
```

```
for x in range(Numero de columnas de la matriz):
    … Recorremos la matriz y volvemos a comprobar si la columna en la que estamos es de
    todos 0s. Si es así entonces:
        for b desde la ultima columna, hasta la columna que hemos encontrado de 0s:
            for a recorriendo todas las filas:
                self.mat[a][b]=self.mat[a][b+1]
#Este proceso se repite tantas veces como columnas ya que se podría dar el caso de varias
columnas de 0s consecutivas que originen que alguna se salte.
```

```
for j desde la ultima columna hasta el numero de columnas menos las de solo 0s:
    for i recorriendo todas las filas:
        self.mat[i][j]=0
#Podría verse como que recorremos la matriz de forma inversa llenando la de 0s, hasta que hemos
llenado tantas columnas de 0s como había antes de empezar.
```

Método **comprobar**:

```
#parámetros iniciales: coordenadas en la matriz y el valor de la bola original seleccionada.
if(self.mat[i][j]!=0):          #Lo primero es comprobar que el sitio seccionado no se un hueco.

    if(j+1<len(self.mat[0]) & valor==self.mat[i][j+1]):     #Comprueba las 4 posiciones
        self.mat[i][j]=0                                    #adyacentes y si alguna es una
        self.comprobar(i, j+1, valor)                       #bola del mismo valor
        self.mat[i][j+1]=0                                  #llama al método comprobar
                                                            #con los nuevos datos.

    if(j-1 >= 0 & valor==self.mat[i][j-1]):
        self.mat[i][j]=0
        self.comprobar(i, j-1, valor)
```

```
                self.mat[i][j-1]=0

        if(i+1<len(self.mat[0]) & valor==self.mat[i+1][j]):
                self.mat[i][j]=0
                self.comprobar(i+1, j, valor)
                self.mat[i+1][j]=0

        if(i-1 >= 0 & valor==self.mat[i-1][j]):
                self.mat[i][j]=0
                self.comprobar(i-1, j, valor)
                self.mat[i-1][j]=0
```

Metodo **point**:

```
x=0             #variable para almacenar el numero de bolas restantes.
for con i recorremos las filas de self.mat:                     #Recorremos la matriz y
        for con j recorremos las columnas de self.mat:          #contamos el numero de
                ...                                             #bolas restantes.
self.puntos += (x-self.nceros)*(x-self.nceros)*5    #suma a los puntos actuales la cantidad
self.nceros = x                                     #establecida por eliminar la diferencia de
                                                    #bolas entre x y self.nceros
```

Método **fin**:

#Se utiliza para comprobar si la partida llego a su fin. Parámetros: modo en el que estamos y el objeto Ranking que tiene las puntuaciones y sus metodos.

```
end=1
for con i recorremos las filas de self.mat:                     #Recorremos la matriz y
        for con j recorremos las columnas de self.mat:
                #recorremos la matriz y para cada bola distinta de 0 comprobamos si tiene alguna
                #del mismo valor que ella adyacente.
                Si hay alguna jugada aun end = 0

if(end==1):
        print Mensaje de no hay mas movimientos.
        x=0
        for con i recorremos las filas de self.mat:                     #Recorremos la matriz y
                for con j recorremos las columnas de self.mat:          #Contamos numero de
                        if(self.mat[i][j]!=0): x+=1                     #bolas restantes.

        if(2000- (x*x*10)>=0):
                self.puntos += (2000-(x*x*10))
```

#Si la puntuación final es record lo muestra por pantalla y la añada en su campo dentro del objeto Ranking y si no es mejor que el record también informa del resultado.

Método **jugada**:

#Parámetros de entrada modo y objeto Ranking, utilizados para pasarlos como parámetros en la función mostrar.

while(Las coordenadas no sean correctas):
    cord=**raw_input**("Introduce las coordenadas:")
    try:
        Si las coordenadas introducidas son correctas, no se vuelve a entrar en el bucle si no se le vuelven a pedir al usuario.
        Si los datos introducidos son 0 0 se la función devuelve un 0 el usuario dese salir.
    except:
        El valor introducido no es valido por lo tanto se vuelven a pedir de nuevo los datos.

```
valor=self.mat[i][j]                    #Se llaman a las funciones necesarias en
self.comprobar(i, j, valor)             #cada jugada para eliminar bolas, y
self.cerosup()                          #realizar los desplazamientos necesarios.
self.columns()
self. point()
self.mostrar(modo, mispuntos)
return 1
```

Método **partida**:

```
self.puntos = 0                 #Al iniciar nueva partida se restablecen los valores de puntos
self.nceros = 0                 #y nceros a 0 para comenzar la nueva partida.
self.mostrar(modo, mispuntos)

end=0                           #Se llama a la función jugada hasta terminar o con la
while(end == 0):                #opción 00 para forzar la salida.
    x = self.jugada(modo, mispuntos)
    end = self.fin(modo, mispuntos)
    if(x==0): end = 2

if(end == 1):
    print "Puntuacion final ..."                    #al terminar la partida por el
    while(Comprobar si quiere jugar una nueva partida):   #método habitual se pregunta
        Si repetir partida return 1                 #si jugar o no una nueva
        Si no repetir partida return 0              #partida.
else: return 0
```

Archivo rank:

Archivo rank.py contiene a la clase Ranking del juego.
Cuenta con una lista bidimensional de 7 listas formadas por dos campos el primero que nos da el nombre del modo y el segundo la puntuación máxima de dicho modo que por defecto comienza en 0.

Método **inicializador (__init__)**:

```
self.listapuntos = […]          #lista por defecto con los modos y las puntuaciones a 0.

try:                            #Se intentar acceder al archivo para leer las puntuaciones.
```

```
        fread = open ("puntuaciones.txt")
        for i recorriendo los modos de listapuntos.          #se intenta acceder a cada puntuación.
                try:
                        self.listapuntos[i][1] = int(fread.readline())
                except:
                        self.listapuntos[i][1] = 0           #si no se encuentra puntuaciones se
except:                                                      #dejan a 0 por defecto.
        print "No se encontró archivo."      #si no se puede acceder al archivo se informa por
                                             #pantalla
```

Método **mostrar**:

```
print "El Ranking de las mejores puntuaciones es:"      #Se muestran por pantalla
for i recorriendo todos los modos:                      #la lista con los records
        for j recorriendo las puntuaciones de cada modo:    #de los modos de juego.
                print self.listapuntos[i][j]
```

Metodo **borrar**:

```
for i recorremos los modos de listapuntos:      #Recorre las puntuaciones
        self.listapuntos[i][1] = 0              #y las pone a 0.
print "Se han borrados las puntuaciones."
```

Metodo **guardar**:

```
fwrite = open ("puntuaciones.txt", "w")     #Recorre las puntuaciones
for i recorremos los modos de listapuntos:  #y las escribe en el
        puntos = str(self.listapuntos[i][1])    #fichero de puntuaciones.
        fwrite.write(puntos)
fwrite.close()
```

Metodo **niveloculto**:

```
activar = True                                          #Comprueba que en los
for i recorremos los modos de listapuntos menos el ultimo:      #distintos modos la
        if(self.listapuntos[][]==0):                    #puntuación sea distinta
                activar = False                         #de 0
return activar
```