```
- [Parte 1. Ejercicios de traducción (4 puntos)](#parte-1-ejercicios-
de-traduccin-4-puntos)
                - [imagen_set_pixel:](#imagensetpixel)
                - [imagen_clean:](#imagenclean)
                - [imagen_init:](#imageninit)
                - [imagen_copy:](#imagencopy)
                - [imagen_dibuja_imagen:](#imagendibujaimagen)
                - [imagen_dibuja_imagen_rotada:]
(#imagendibujaimagenrotada)
                - [nueva_pieza_actual:](#nuevapiezaactual)
                - [intentar_movimiento:](#intentarmovimiento)
                - [intentar_rotar_pieza_actual:]
(#intentarrotarpiezaactual)
                - [bajar_pieza_actual:](#bajarpiezaactual)
- [Parte 2. Ejercicios de implementación (6 puntos)](#parte-2-
ejercicios-de-implementacin-6-puntos)
                - [Marcador de puntuación:](#marcador-de-puntuacin)
                - [Final de la partida:](#final-de-la-partida)
                - [Completando líneas:](#completando-lneas)
                - [Eliminando líneas:](#eliminando-lneas)
                - [Ritmo de caída:](#ritmo-de-cada)
```

##Parte 1. Ejercicios de traducción (4 puntos)

En general esta primera parte no ha tenido casi dificultad, ya que solo era pasar codigo de C a mips. La parte mas problematica ha sido la de hacer las llamadas a las funciones de forma correcta para asegurarse de que estaban correctamente implementadas.

La forma de realizar la practica ha sido: primero hemos ido portando todo el codigo a mips, una vez que se porto todo el codigo se fue comprobando cada funcion que estaba correctamente hecha y corrigiendo las pequeñas erratas que pudiesen tener.

imagen_set_pixel:

Realizar esta funcion ha sido facil ya que era igual que `imagen_get_pixel` con la pequeña diferencia que habia que guardar el color.

imagen_clean:

Para la complejidad de esta funcion ha sido que nunca habiamos realizado un for dentro de otro for, pero con un pequeño dibujo hemos podido implementarlo correctamente a la primera (Para todos los demas doble for que hay en el juego hemos copiado el codigo de este).

La llamada a la funcion para comprobar si funcionaba correctamente ha sido:

```
la $a0, pieza_actual
li $t0, 9
sw $t0, 0($a0)
li $t0, 5
sw $t0, 4($a0)
li $a1, '+'
jal imagen_clean
la $a0, pieza_actual
jal imagen_print
```

imagen_init:

Esta funcion no ha tenido ninguna complejudad.

La llamada a la funcion para comprobar si funcionaba correctamente ha sido:

```
la $a0, pieza_actual
li $t0, 30
sw $t0, 0($a0)
li $t0, 30
sw $t0, 4($a0)
li $a1, 8
li $a2, 4
li $a3, '*'
jal imagen_init
la $a0, pieza_actual
jal imagen_print
```

imagen_copy:

Partiendo de la base de que ya tenemos hecho el doble for, implementar el resto ha sido facil.

La llamada a la funcion para comprobar si funcionaba correctamente ha sido:

```
la $a0, pieza_actual    #dst
la $a1, pieza_jota      #src
jal imagen_copy
la $a0, pieza_actual
jal imagen_print
```

imagen_dibuja_imagen:

Lo unico que nos ha costado era que no teniamos claro como poner la constaste `PIXEL_VACIO`, por lo que hemos mirado como estaba implementada en la funcion `jugar_partida` y hemos visto que era un simple *0*.

La llamada a la funcion para comprobar si funcionaba correctamente ha sido:

```
la $a0, pieza_actual    #dst
li $t0, 23
sw $t0, 0($a0)
sw $t0, 4($a0)
la $a1, pieza_ele       #src
li $a2, 8
li $a3, 8
jal imagen_dibuja_imagen
la $a0, pieza_actual
jal imagen_print
```

imagen_dibuja_imagen_rotada:

No ha tenido ninguna dificultad, el codigo era basicamente el mismo, solo habia que hacer unas sumas y restas en algunos parametros de la funcion `imagen_set_pixel`

La llamada a la funcion para comprobar si funcionaba correctamente ha sido:

```
la $a0, pieza_actual    #dst
li $t0, 23
sw $t0, 0($a0)
sw $t0, 4($a0)
la $a1, pieza_ele       #src
li $a2, 8
li $a3, 8
jal imagen_dibuja_imagen_rotada
la $a0, pieza_actual
jal imagen_print
```

nueva_pieza_actual:

El problema que hemos tenido con esta practica era que para modificar el valor de `pieza_actual_x` estabamos usando un `lw` en vez de `la`, una vez que nos dimos cuenta de ese fallo el resto funciono correctamente.

La llamada a la funcion para comprobar si funcionaba correctamente ha sido:

```
jal nueva_pieza_actual
la $a0, pieza_actual
jal imagen_print
```

intentar_movimiento:

En esta tuve que corregir el mismo fallo que en `nueva_pieza_actual` por usar `lw pieza_actual_x`

A partir de aqui ya no tuve que implemetar yo codigo para llamar a la funcion, ya el tetris puede funcionar con el codigo del main

intentar_rotar_pieza_actual:

Al probar intentar_movimiento comprobe que ya era totalmeten funcional el tetris y que podia rotar, mover a derecha e izquierda y baja la pieza, por lo que no me hizo falta revisar esta funcion

bajar_pieza_actual:

Al probar intentar_movimiento comprobe que ya era totalmeten funcional el tetris y que podia rotar, mover a derecha e izquierda y baja la pieza, por lo que no me hizo falta revisar esta funcion

##Parte 2. Ejercicios de implementación (6 puntos)

Marcador de puntuación:

El principal problema de esta funcion ha sido entender exactamente como habia que implementarlo, ya que no entiendia bien como hacer la función `imagen_dibuja_cadena`, pensaba que tenia que llamar a la funcion `print_character` en vez de a `imagen_set_pixel`. Una vez coseguido hacer esta funcion correctamente inicializar el marcador y actualizarlo ha sido cosa trivial, tampoco ha sido complejo usar la funcion `integer_to_string`.

Final de la partida:

Completando líneas:

Eliminando líneas:

Ritmo de caída:

Esta función no es dificil de implementar, solamente hay que hacer que en vez de trabajar con un numero fijo trabajes con una variable global que vas modificando, para hacerlo mas facil he implementado un procedimiento llamado `calcula_tiempo` que sin tener que pasarle nada coje el valor del tiempo calcula el 10% y se lo resta, despues lo actualiza. Ya que asi es mas facil de entender el codigo. Una vez hecho esto lo que hay que hacer es en el contador cada vez que sumas un punto calcular si la puntuacion es multiplo de 50, en caso de que lo sea se llama al procedimiento y aumentamos la velocidad de juego. Una cosa que no habiamos pensado y que al repetir el juego varias veces es que en la funcion `jugar_partida`, cuando inicializas el marcador a 0 tambien tienes que inicialalizar el timpo a 1000, ya que sino, el tiempo despues de acabar una partida no se inicializa y mantiene la velocidad de la partida anterior.