Parte 1. Ejercicios de traducción (4 puntos)

En general esta primera parte no ha tenido casi dificultad, ya que solo era pasar código de C a mips.
La parte mas problemática 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 código a mips, una vez que se porto todo el código se fue comprobando cada función que estaba correctamente hecha y corrigiendo las pequeñas erratas que pudiesen tener.

imagen_set_pixel:

Realizar esta función ha sido fácil ya que era igual que `imagen_get_pixel` con la pequeña diferencia que había que guardar el color.

imagen_clean:

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

La llamada a la función 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 función no ha tenido ninguna complejidad.

La llamada a la función 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 fácil.

La llamada a la función 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 único que nos ha costado era que no teníamos claro como poner la constaste `PIXEL_VACIO`, por lo que hemos mirado como estaba implementada en la función `jugar_partida` y hemos visto que era un simple *0*.

La llamada a la función 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 código era básicamente el mismo, solo había que hacer unas sumas y restas en algunos parámetros de la función `imagen_set_pixel`

La llamada a la función 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` estábamos usando un `lw` en vez de `la`, una vez que nos dimos cuenta de ese fallo el resto funciono correctamente.

La llamada a la función 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 aquí ya no tuve que implementar yo código para llamar a la función, ya el tetris puede funcionar con el código del main

intentar_rotar_pieza_actual:

Al probar intentar_movimiento comprobé que ya era totalmente funcional el tetris y que podía rotar, mover a derecha e izquierda y baja la pieza, por lo que no me hizo falta revisar esta función

bajar_pieza_actual:

Al probar intentar_movimiento comprobé que ya era totalmente funcional el tetris y que podía rotar, mover a derecha e izquierda y baja la pieza, por lo que no me hizo falta revisar esta función

Parte 2. Ejercicios de implementación (6 puntos)

Marcador de puntuación:

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

Final de la partida:

Esta implementación ha sido fácil de implemente, solo hay que poner un `if` en la función `bajar_pieza_actual` para comprobar si puedes

volver hacer un movimiento con la pieza_actual.
El único problema que hemos tenido es que al generar una nueva pieza, se genera en la misma coordenada `y` de donde se puso la anterior, y si esa posición `y` estaba en un lateral el programa podía detectar que no podía bajar la ficha y llamaba a `acabar_partida`

```
move    $a0, $s0            #Mal
lw    $a0, pieza_actual_x        #Bien
```

Completando líneas:

Comprobar si habíamos completado una/varias linea/as no ha sido complejo, solo eran 2 bucles for anidados, que para cada posición `y` de la altura de la pieza teníamos que comprobar si había algún pixel vació en toda la anchura del campo, silo había actualizábamos el marcador con un +10.

Eliminando líneas:

Esta función ha sido mas compleja de implementar, ya que estábamos haciendo un bucle for ascendente en vez de descendente, por lo que aunque eliminábamos la linea que deseábamos la ultima no eramos capaces de ponerle un `'\0'` para eliminarla también.

Ritmo de caída:

Esta función no es difícil 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 fácil he implementado un procedimiento llamado `calcula_tiempo` que sin tener que pasarle nada coge el valor del tiempo calcula el 10% y se lo resta, después lo actualiza. Ya que así es mas fácil de entender el código. Una vez hecho esto lo que hay que hacer es en el contador cada vez que sumas un punto calcular si la puntuación es múltiplo de 50, en caso de que lo sea se llama al procedimiento y aumentamos la velocidad de juego.
Una cosa que no habíamos pensado y que al repetir el juego varias veces es que en la función `jugar_partida`, cuando inicializas el marcador a 0 también tienes que inicialalizar el tiempo a 1000, ya que sino, el tiempo después de acabar una partida no se inicializa y mantiene la velocidad de la partida anterior.

1.5.3 Funcionalidad opcional (hasta 1,5 puntos adicionales)

Configuración:

Para implementar esta función lo primero ha sido añadir a las librerías del sistema las funciones:

- read_integer
- print_integer

Después hemos añadido al menú la opción de llamar a la función para editar la configuración, esta funciona es un `do .. while` que mostrara un menú similar al del tetris que se usara para llamar a las funciones encargadas de editar las diferentes opciones.
Para poder comparar que los valores del `campo` no sean nunca superiores a los de `pantalla` los hemos puesto globales.

Las funciones que modifican los valores son similares todas, limpian la pantalla, piden el valor a actualizar, comprueban que es valido tanto al alza como a la baja y en caso de que sea valido lo guardan.

También ha habido que modificar el valor de `pieza_actual_x` para que sea `campo_x div 2`, ya que sino no sale centrada cuando cambias el valor del campo

Para la modificación de de las teclas solo ha habido que saber en que posición estaban guardadas del diccionario `procesar_entrada.opciones`, y una vez localizadas las posiciones son `sb` introducir las nuevas teclas.