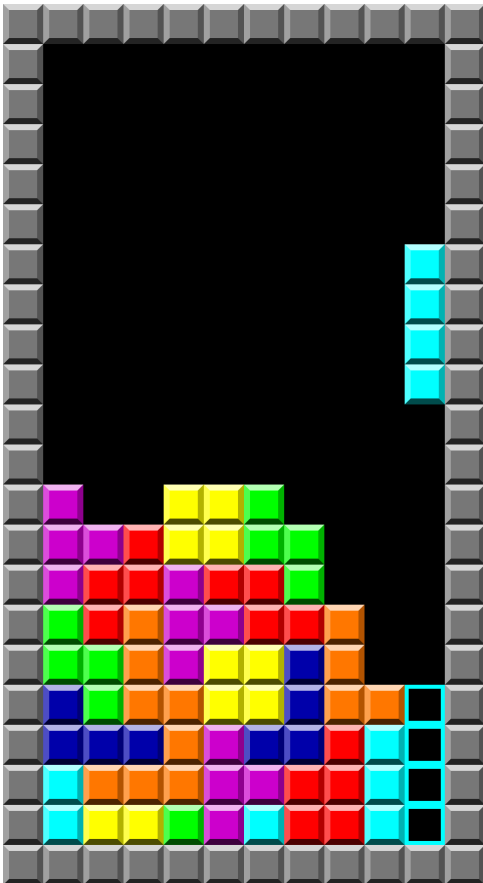


Práctica: TETRIS

Juanan Pereira

27 de febrero de 2021



Índice

1. Introducción	1
1.1. Reglas básicas	1
1.2. Interacción del usuario	1
1.3. Diseño de la Práctica	1
2. Visión General de las Clases del Tetris	2
2.1. Clases básicas	2
2.2. Puntos	2
2.3. Rectángulos	2
2.4. Bloques	2
3. El juego del Tetris	3
3.1. Tetromino I	3
3.2. Más Tetrominos	4
4. Crear una pieza al azar	5
5. Eventos de teclado	5
6. Mover las piezas	6
7. ¡Piezas fuera!	6
8. Añadir una pieza al tablero	7
8.1. Bajar pieza con tecla de barra espaciadora	8
9. Colisiones con otras piezas	8
10. Rotar una Pieza	8
11. Cómo mover <i>automáticamente</i> las piezas	9
12. Eliminar líneas completas	10
13. Game Over	11
14. Bonus points (Opcional)	11
14.1. Puntuación y niveles	11
14.2. Previsualización de la siguiente Pieza	11
14.3. Pausar el juego	12
14.4. Música y efectos sonoros	12
14.5. Récores	12
14.6. Pruebas unitarias con QUnit desde consola	12
14.7. Refactorización del método de pintar en pantalla	12
14.8. Dos jugadores	13
14.9. Juego en red	13

1. Introducción





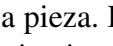
El juego del Tetris es considerado como uno de los videojuegos más populares de todos los tiempos. Se basa en un juego tipo puzle desarrollado por Alexey Pajitnov en 1984 mientras trabajaba en la Academia de las Ciencias (Moscú) de la antigua Unión Soviética. Desde entonces, se han desarrollado cientos de variantes del juego. En esta práctica, vamos a crear nuestra propia versión del Tetris, descrita en los siguientes párrafos.

El juego comienza con un tablero vacío. El tablero se compone de 10x20 casillas. La casilla de la esquina superior izquierda tiene coordenadas (0,0) y la casilla de la esquina inferior derecha tiene coordenadas (9,19) (como ves, el eje y crece hacia abajo). Se elige una pieza al azar de las 7 posibles piezas del Tetris y se pinta en la parte superior del tablero. La pieza comienza a caer a intervalos regulares, una casilla cada vez.

1.1. Reglas básicas

1. La pieza no puede caer en una casilla ocupada por otra pieza o fuera de los límites del tablero.
2. Cuando la pieza que cae se topa con otra -o con la parte inferior del tablero-, deja de moverse. En ese momento, otra nueva pieza comienza a caer de la parte superior del tablero.
3. A medida que las piezas van cayendo en la parte inferior, comienzan a generarse filas de piezas. Si la fila es completa (tiene rellenas todas las casillas que la forman), desaparecerá del tablero y todo los bloques por encima de la misma caerán hasta toparse con otros bloques o con el fondo del tablero.
4. Cuando no pueda dibujarse una nueva pieza en la parte superior del tablero (porque hay demasiadas filas debajo), el juego termina (*Game Over*).

1.2. Interacción del usuario

El usuario puede usar las flechas del teclado para rotar y mover las piezas. Las flechas 'Izquierda' , 'Derecha' , 'Abajo'  mueven la pieza una casilla hacia esa dirección. La flecha 'Arriba'  rota la pieza. El usuario también puede pulsar la barra espaciadora () lo que hará que la pieza caiga inmediatamente hasta que o bien se tope con otra pieza o bien se tope con el borde inferior del tablero. Al mover o rotar una pieza, ésta no puede solaparse con otra o salir fuera de los límites del tablero.

1.3. Diseño de la Práctica

Para ayudarte con el diseño de la práctica, dispones de un fichero de ayuda (plantilla en JavaScript) con las clases que tendrás que programar, junto con las definiciones de sus métodos (que tendrás que implementar). Iremos paso a paso, de forma incremental, probando en todo momento los métodos y clases que vayas implementando.

Fíjate que todos los métodos tienen un comentario 'AQUÍ TU CÓDIGO'. Debes borrar dicho comentario e implementar en su lugar cada método en cuestión. Para que el Tetris funcione deberás cumplimentar todos los métodos que se pedirán en cada enunciado.

2. Visión General de las Clases del Tetris

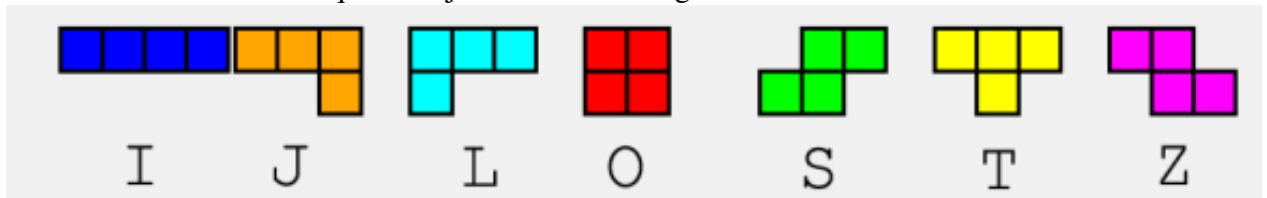
Descarga una copia del fichero `tetris.js` de la web del curso y échale un vistazo. Además de las clases básicas **Point**, **Rectangle**, **Block** y **Shape** que implementaremos a continuación para conseguir disponer de las piezas del Tetris, hay dos clases más: **Board** y **Tetris**. La clase **Board** implementará la funcionalidad del tablero. La clase **Tetris** implementará el controlador del juego.

2.1. Clases básicas

Inicialmente, construiremos las piezas que formarán parte del juego. Estas piezas se conocen con el nombre de tetrominos.

Cada uno de los tetrominos tiene 4 bloques; un bloque es un cuadrado de color. Observa que la relación 'tiene' realmente significa contiene, se implementa con un atributo contenedor, mientras que la relación 'es', implica una relación de herencia.

Los 7 tetrominos con los que trabajaremos son los siguientes:



Para poder construir un tetromino debemos implementar primero las clases básicas: **Point**, **Rectangle** y **Block**.

2.2. Puntos

Point es una clase básica que se limita a representar la posición (x,y) de un punto en pantalla. Está programada en la plantilla (la primera clase ha sido fácil, ¿no?)

2.3. Rectángulos

Rectangle es realmente la primera clase que tendrás que programar, pero tendrás una ayuda, dado que algunos métodos ya están programados: el método constructor, el método de inicialización y los métodos para asignar la anchura de línea y el color de fondo del rectángulo. Tu labor consiste en programar el resto de los métodos siguiendo los comentarios que verás en el código fuente y empezando por el método `draw`.

2.4. Bloques

Como se ha indicado, el tablero típico del Tetris se compone de 10x20 casillas, donde una casilla es un cuadrado de 30 pixels de ancho. Cada bloque ocupa una única casilla. En este ejercicio, representaremos el tablero en términos de cuadrados (no de pixels) - por lo que cuando hablemos de puntos del tablero estaremos dando posiciones de las casillas. Lógicamente, para poder pintar los bloques en pantalla, deberás implementar métodos que sepan pasar de una posición-casilla a una posición-pixel.

Crea una nueva clase **Block** que herede de la clase **Rectangle**¹. Debe tener atributos `x` e `y` que correspondan a la posición del bloque en un tablero del Tetris, en términos de casillas, no

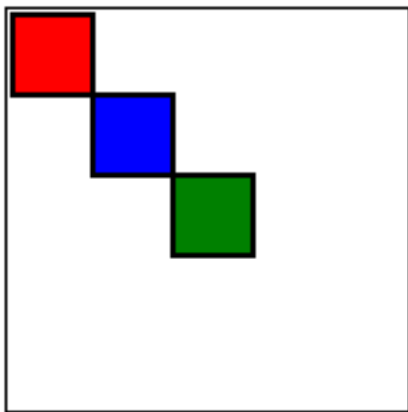
de pixels. La posición (0,0) es la esquina superior izquierda del tablero y la posición (9,19) es la esquina inferior derecha.

Un ejemplo de cómo usar el bloque (una vez construidas las clases `Point`, `Rectangle` y `Block` solicitadas):

```
1 var block1 = new Block(new Point(0,0), 'red'),
2   block2 = new Block(new Point(1,1), 'blue'),
3   block3 = new Block(new Point(2,2), 'green');
4
5 block1.draw();
6 block2.draw();
7 block3.draw();
```

Fíjate que el constructor de `Block` debe aceptar dos parámetros: un punto que especifica la casilla en la que colocar el bloque y el color del bloque.

Y así se verá en un tablero de 5x5:



Implementa los métodos que se han indicado y comprueba que tu aplicación pasa las pruebas unitarias del ejercicio: <http://jsfiddle.net/AinhoaY/hq7Lt1do/5/>.

3. El juego del Tetris

3.1. Tetromino I

Crea una clase para representar una pieza y llámale `Shape`. Como hemos visto, una pieza debe contener una *lista de bloques* como atributo, y un método `draw` para pintar la pieza en pantalla. Una vez implementada, añade esta clase `I_Shape` a tu código. Esta clase debe heredar de la clase `Shape`.

```
1 function I_Shape(center) {
2   var coords = [new Point(center.x - 2, center.y),
3                 new Point(center.x - 1, center.y),
4                 new Point(center.x , center.y),
5                 new Point(center.x + 1, center.y)];
6 }
```

¹Se ha decidido usar nombres de clases, objetos, atributos y métodos en inglés, evitando problemas de acentos, ñes y de nombres de métodos en *spanglish* como `setAnchura`, `getAltura`, etc.

```

7 | Shape.prototype.init.call(this, coords, "blue");
8 | }

```

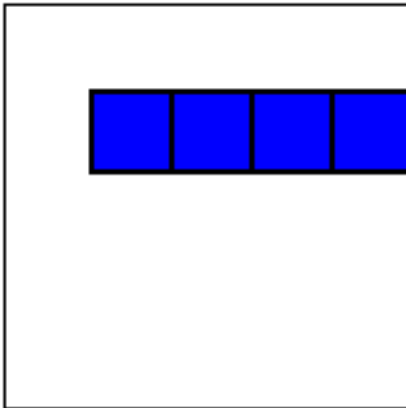
El parámetro `center` es un `Point` que guarda la posición del bloque central de la pieza². Verifica que tu código muestra la figura `I_Shape` correctamente. Aquí tienes un ejemplo:

```

1 | var shape = new I_Shape(new Point(3, 1));
2 | shape.draw();

```

El resultado en pantalla debería ser el siguiente:



Fíjate que en el código anterior, el tercer bloque de la pieza `I_Shape` corresponde a la posición (3,1).



Implementa los métodos que se han indicado y comprueba que tu aplicación pasa las pruebas unitarias del ejercicio: <http://jsfiddle.net/AinhoaY/pgmdq86j/15/>.

3.2. Más Tetrominos

A continuación, deberás crear subclases para el resto de las figuras. Fíjate que cada pieza tiene un bloque marcado como el bloque central (aunque en pantalla no debe verse en negro, se muestra aquí en ese color por claridad).



El siguiente código permite dibujar en pantalla todas las piezas definidas en el punto anterior (como se ha indicado, todos los bloques de una pieza deben tener el mismo color - el color negro se muestra como aclaración para conocer cuál debería ser el bloque central en cada caso, no debe aparecer en pantalla):

²La pieza rotará sobre el bloque central cuando implementemos la rotación.

```

1 // una lista con los 7 tetrominos. Debes crear las 6 clases que te faltan
2
3 var tetrominoes = [I_Shape, J_Shape, L_Shape, O_Shape, S_Shape, T_Shape,
4   Z_Shape];
5 x = 3;
6 tetrominoes.forEach( function ( tetromino ) {
7   shape = new tetromino(new Point(x, 1));
8   shape.draw();
9   x += 4;
10 });

```

Nota: los colores que se muestran en la figura anterior son : blue, orange, cyan, red, green, yellow y magenta, respectivamente.



Implementa los métodos que se han indicado y comprueba que tu aplicación pasa las pruebas unitarias del ejercicio: <http://jsfiddle.net/AinhoyaY/burfwkpa/12/>.

4. Crear una pieza al azar

Hemos creado ya los tetrominos que forman parte del núcleo del juego. Éstos pueden pintarse en pantalla, dentro de un elemento canvas que simulará el tablero del Tetris. A partir de esta práctica comenzaremos a implementar la *lógica* del juego.

Implementa el método `create_new_shape` de la clase `Tetris`, que crea y devuelve una nueva pieza elegida al azar. Fíjate que deberás usar el atributo `SHAPES` (que contiene una lista de clases `Shape`).

Más adelante, necesitarás tener una referencia a la pieza que has creado para poderla mover o rotar. La clase `Tetris` guarda para ello el atributo `current_shape`. Modifica el método `Tetris.init` para que muestre la `current_shape` en el tablero.



Implementa los métodos que se han indicado y comprueba que tu aplicación pasa las pruebas unitarias del ejercicio: <http://jsfiddle.net/AinhoyaY/h8ux2L19/34/>. Ejecuta tu código y asegúrate de que ves la pieza en pantalla.

5. Eventos de teclado

Antes de poder mover las piezas en pantalla, necesitaremos aprender a gestionar los eventos de teclado, es decir, conocer cuándo el usuario ha pulsado una tecla.

Debes añadir un gestor de pulsaciones de tecla (`EventListener`) al método `Tetris.init`. Este gestor debe llamar al método `key_pressed` cuando el usuario pulse cualquier tecla.

Por el momento, dicho método debe mostrar por consola un mensaje indicando la tecla que se ha pulsado.

No hay pruebas unitarias para este ejercicio, pero no pases al siguiente hasta no terminar este.

6. Mover las piezas

Observa que los métodos `move` de las clases `Shape`, `Block` y `Rectangle` vienen ya programados en la plantilla, al igual que el método `Rectangle.erase` que borra un bloque del tablero (se borra la posición actual y se pinta el bloque en la nueva posición, simulando movimiento de la pieza).



Atención

Para esta sección de la práctica debes modificar temporalmente el método `Tetris.create_new_shape` para que devuelva siempre una `S_Shape`. Esto lo hacemos así para que las pruebas unitarias sean más sencillas de implementar. Una vez finalizado este ejercicio, debes dejar el código de `Tetris.create_new_shape` tal y como lo tenías.

Una vez controlados los eventos de teclado, modifica los métodos `Tetris.key_pressed` y `Tetris.do_move` para hacer que las piezas se muevan cuando el usuario pulse `←`, `→` y `↓`. Échale un vistazo al atributo `DIRECTION` de la clase `Tetris`. Es un array asociativo con una clave de tipo `string` que especifica la dirección del movimiento de la pieza, y un valor `(dx,dy)` que corresponde al número de unidades que debe moverse en el eje `x` e `y` respectivamente.

¡No trates de implementar toda la funcionalidad del método `Tetris.do_move`! -añadiremos las cosas que faltan más adelante. Por ahora, añade simplemente el código para mover la pieza en la dirección apropiada especificada por el parámetro `direction`.



Implementa los métodos que se han indicado y comprueba que tu aplicación pasa las pruebas unitarias del ejercicio: <http://jsfiddle.net/AinhoaY/beun98tm/30/>.

7. ¡Piezas fuera!

¿Qué ocurre si mueves la pieza a la izquierda 100 veces? Como puedes observar, la pieza desaparece por completo por la parte izquierda del tablero. Debemos asegurarnos de que esto no ocurra. Las piezas no deben salir fuera de los límites del tablero. Modifica tu código de tal forma que una pieza sólo se mueva si ese movimiento es posible, i.e. si alguno de sus bloques se saliera del tablero la pieza en su conjunto no debe moverse. Para ello:

1. Modifica el método `Board.can_move` para que no devuelva siempre `true` (como hace ahora) sino que compruebe que la posición que se le pasa como parámetro esté dentro de los límites del tablero. Devuelve `true` en caso afirmativo y `false` en caso contrario.
2. Modifica el método `Block.can_move` - para que llame al método anterior con la posición actual incrementada en `dx, dy`. Observa que `dx,dy` llegan como parámetro.
3. Modifica ahora el método `Shape.can_move`. Devuelve `true` si la pieza puede moverse `dx,dy` unidades (el incremento le llega como parámetro) y `false` en caso contrario. (Pista: este método utiliza el método `Block.can_move` que acabas de escribir, llamándolo una vez por cada bloque de los que forman la pieza).

4. Para finalizar, actualiza el método `Tetris.do_move`, que antes de mover una pieza, comprueba si este movimiento es posible.



Ejecuta tu código y asegúrate de que las piezas no caen fuera de los límites del tablero (por la izquierda, derecha o por abajo). Comprueba que tu aplicación pasa las pruebas unitarias del ejercicio: <http://jsfiddle.net/AinhoaY/jf8pqu6/43/>

8. Añadir una pieza al tablero

Ahora que las piezas ya no se pierden por debajo del tablero, continuemos con el juego. Una vez que la pieza toca el borde inferior del tablero, debería ser añadida de forma permanente al juego y aparecer una nueva pieza en la parte superior. ¿Cómo saber que la pieza ha tocado el borde inferior? ¿Cómo añadir la pieza al tablero? ¿Qué estructura de datos podría albergar las piezas permanentes del tablero?

Para gestionar el estado del tablero y las piezas permanentes usaremos el atributo `grid` del objeto `Board`. La rejilla `grid` es un array asociativo (también conocido como diccionario) donde la clave es una tupla `(x,y)` - representada en forma de string- que corresponde a la posición `(x,y)` del tablero. El valor de esta clave será un objeto `Block` correspondiente a esa casilla del tablero.

Modifica tu código para que añada la pieza al tablero y a continuación instancie una nueva, situándola en la parte superior. Para ello:

1. Modifica el método `Board.add_shape` para que añada cada bloque al diccionario `grid`.³
2. Actualiza el método `Tetris.do_move` de tal forma que si el último movimiento que falló (que no pudo ejecutarse porque la pieza se salía de los límites del tablero) fue 'Down', el método haga lo siguiente:
 - añadir la pieza actual (`current_shape`) al tablero,
 - actualizar el valor del atributo `Tetris.current_shape` con una nueva pieza al azar,
 - dibujar la nueva pieza en el tablero.



Ejecuta tu código y asegúrate de que cuando una pieza alcanza el borde inferior se añade al tablero y se pinta una nueva pieza (elegida al azar) en la parte superior. Debes pasar las siguientes pruebas unitarias: <http://jsfiddle.net/AinhoaY/na6yzex2/76/>

³Actualmente `Shape.blocks` es un atributo público, pero idealmente debería encapsularse y crear un método `getter` de acceso.

8.1. Bajar pieza con tecla de barra espaciadora

Actualiza el método `Tetris.key_pressed` para que la pieza caiga de forma inmediata hacia el borde inferior del tablero si el usuario pulsa `Espacio`.



Ejecuta las siguientes pruebas unitarias para asegurarte de que tu código es correcto: <http://jsfiddle.net/AinhoaY/htu6zcr9/40/>

9. Colisiones con otras piezas

¿Qué ocurre cuando una pieza trata de moverse a una casilla ocupada por otra pieza? ¿Cómo modificarías tu código para asegurarte de que una pieza no puede moverse a una casilla ya ocupada?

Modifica el método `Board.can_move` para comprobar si existe ya una pieza en la posición actual. Debe devolver `true` si la casilla está vacía y `false` en caso contrario. Pista: usa el operador `in` en el diccionario `grid` para comprobar si hay un valor (bloque) en la clave (x,y).



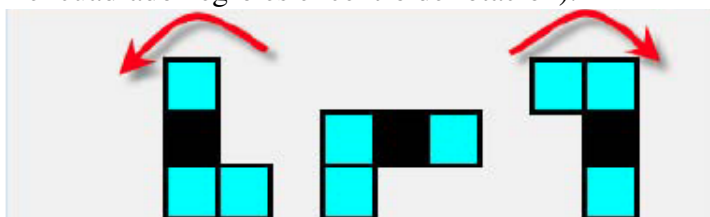
Ejecuta tu código y asegúrate de que las piezas no se solapan. Comprueba que tu código pasa la siguiente prueba unitaria: <http://jsfiddle.net/AinhoaY/ac20f4yp/76/>

10. Rotar una Pieza

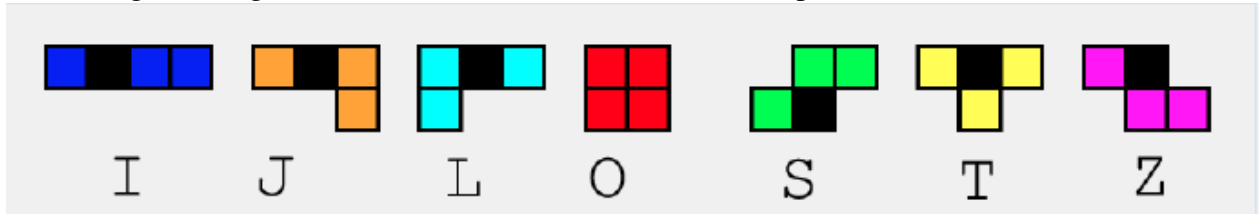
Mover una pieza no ha sido difícil pero, ¿cómo podemos rotarla? Para ello, es necesario conocer la fórmula que nos permite rotar una casilla alrededor de otra un ángulo de 90 grados:


```
x = center.x - dir*center.y + dir*block.y
y = center.y + dir*center.x - dir*block.x
```

Esta fórmula nos da las nuevas coordenadas del objeto `block` de tipo `Bloque`, cuando se rota alrededor del otro objeto `center` (también de la clase `Block`). La variable `dir` especifica la dirección de rotación (puedes obtener la dirección actual de rotación consultando el atributo `Shape.rotation_dir` que hemos añadido en la inicialización de `Shape`). Si `dir = 1`, el bloque se rota en el sentido de las agujas del reloj y si `dir=-1`, en sentido anti-horario. En la siguiente figura, el cuadrado negro representa el bloque `center` y el resto de los bloques rotan con respecto a él (es decir el cuadrado negro es el centro de rotación).




Hay que tener en cuenta un detalle: no todas las piezas se comportan igual. En concreto, las piezas J, L y T rotan siempre en sentido horario. Las piezas I, S y Z oscilan (primero rotan en sentido horario y luego en sentido anti-horario). La pieza O no rota. Para poder implementar la rotación en cada tipo de pieza necesitas conocer primero cuál es su centro de rotación. El cuadrado negro en la siguiente figura muestra el centro de rotación de cada pieza.



1. Modifica la clase `Shape` para añadir un atributo `shift_rotation_dir` con valor `true` (por defecto, las piezas giran en un sentido y luego en otro, pero como hemos visto, no todas lo hacen. Este atributo indicará cuáles de ellas oscilan y cuáles no. Por defecto, oscilan);
2. Modifica el constructor de las piezas necesarias para guardar en el atributo `center_block` cuál es el bloque central de cada una de ellas (en la figura superior el bloque central está marcado con fondo negro).
3. Implementa los métodos `Shape.can_rotate` y `Shape.rotate`. Las piezas no pueden rotar fuera de los límites del tablero o sobre otras piezas. Así, el método `Shape.can_rotate` debe devolver `false` si alguno de los bloques de la pieza no puede moverse a su nueva posición en el tablero (bien porque queda fuera de los límites o bien porque está ocupada)
4. Implementa ahora el método `Tetris.do_rotate` para rotar la pieza actual si dicha rotación es posible.
5. Para finalizar, modifica el método `Tetris.key_pressed` para rotar una pieza cuando se pulse la tecla .
6. Opcional: tal vez quieras sobrescribir el método `can_rotate` de `O_Shape` para que siempre devuelva `false` (realmente una `O_Shape` no rota).



Ejecuta tu código y asegúrate de que las piezas rotan cuando pulsas la tecla . Asegúrate de que tu código pasa las pruebas unitarias definidas aquí: <http://jsfiddle.net/AinhoaY/gd50oeL3/145/>

11. Cómo mover *automáticamente* las piezas

Crea el código del método `Tetris.animate_shape`. Este método debe ejecutarse de forma automática una vez cada segundo para mover la pieza actual hacia abajo (una casilla). Modifica después el método `Tetris.init` para que comience la animación de la pieza una vez que se dibuje en pantalla.

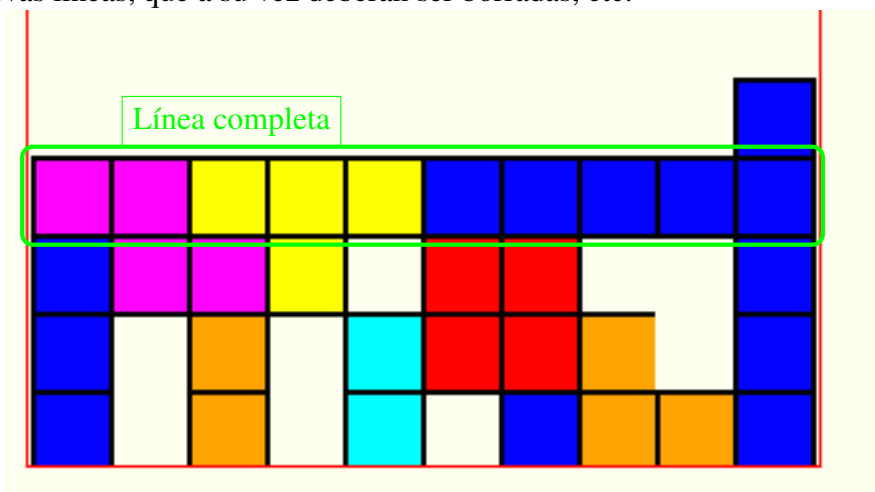


Ejecuta tu código y asegúrate de que las piezas caen por sí solas. Prueba tu código con estas pruebas unitarias: <http://jsfiddle.net/AinhoaY/ds1rLw5f/129/>

Tu juego ahora debería ser casi jugable... ¡casi!

12. Eliminar líneas completas

Cuando se añade una nueva pieza al tablero, es necesario comprobar la existencia de líneas que se hayan podido completar gracias a esta nueva pieza. En caso afirmativo, esa línea debe eliminarse, es decir, todos los bloques que formen esa línea deben borrarse. Una vez borrados, todos los bloques por encima de la línea borrada caerán una casilla hacia abajo. Esto puede provocar que se formen nuevas líneas, que a su vez deberán ser borradas, etc.



La clase `Board` tiene varios métodos para ayudarte a implementar esta funcionalidad - `is_row_complete` (devuelve `true` si todas las casillas de la fila que se le pasa como parámetro están ocupadas), `delete_row` (borra una fila), `move_down_rows` (mueve todas las filas por encima de la fila pasada como parámetro una casilla hacia abajo), y `remove_complete_rows` (comprueba si existe alguna fila completa y la elimina, moviendo todas las filas por encima una casilla hacia abajo).

1. Implementa los cuatro métodos descritos en el párrafo anterior. Pista: recuerda que los diccionarios disponen del método `del`.
2. A continuación, modifica el método `Tetris.do_move` de tal forma que cada vez que una pieza se haya añadido al tablero de forma permanente, compruebe si existe alguna fila completa y en caso afirmativo, las elimine.



Ejecuta tu código y asegúrate que al jugar, las filas completas se eliminan correctamente. Comprueba tu código con las siguientes pruebas unitarias: <http://jsfiddle.net/AinhoaY/ut6L3dyc/49/>

13. Game Over

Quedan algunos toques finales. Antes de colocar una nueva pieza en el tablero, debes comprobar que la posición no esté ya ocupada. Si así fuera (si estuviera ocupada), el juego deberá terminar, mostrar un texto en pantalla (¡*Game Over* es un clásico) y dejar de colocar nuevas piezas. Pista: un buen sitio donde colocar la comprobación de fin de juego es en el método `Tetris.do_move`, justo tras comprobar que no es posible eliminar filas. El método `game_over` debería parar el timer que pusiste en marcha para que las piezas se movieran automáticamente.

1. Implementa el método `Board.game_over` para que muestre el mensaje “Game Over!!!” en pantalla al finalizar el juego.
2. Modifica el método `Tetris.do_move` para que llame al método `game_over` si no es posible colocar más piezas en el tablero.



Ejecuta tu código y asegúrate de que el mensaje *Game Over* aparece en pantalla cuando ya no puedas colocar más piezas.

¡ENHORABUENA! Has terminado de programar tu propio juego del Tetris :-)

14. Bonus points (Opcional)

Si quieres hacer que tu juego sea aún más entretenido, te propongo a continuación algunas funcionalidades que te ayudarán a lograrlo. No hay ayuda en el código de la plantilla para implementar estas sugerencias pero merece la pena intentarlo, ¿no crees?.

14.1. Puntuación y niveles

Modifica tu juego de tal forma que se muestre una puntuación mientras el usuario está jugando. Puedes elegir tu propia estrategia de puntuación. En general, parece razonable puntuar cuando el usuario consigue hacer una línea (y darle más puntos si consigue 2 o más en secuencia). Conseguir cuatro líneas seguidas se llama hacer un Tetris.

También puedes añadir nuevos niveles de juego. Por ejemplo, cuando el usuario supere cierto umbral de puntuación, pasará de nivel. Esto puede significar, por ejemplo, que las piezas caen más rápido. Si implementas esta opción, deberías mostrar por pantalla el nivel actual en todo momento.

14.2. Previsualización de la siguiente Pieza

Para ayudar al usuario a hacer filas, puedes mostrar en pantalla una visualización de cuál será la siguiente pieza que caerá. Tendrás que pensar qué métodos necesitarás para mostrar en pantalla la siguiente pieza y que ésta se convierta en la actual cuando sea necesario.

14.3. Pausar el juego

Modifica tu código para que el jugador pueda pulsar la tecla `p` o `P` cuando quiera pausar el juego. Para volver a jugar deberá pulsar la misma tecla. En modo pausa,

- la pieza deja de caer automáticamente
- el jugador no podrá mover ni rotar la pieza actual
- se mostrará un mensaje en pantalla indicando que el juego está en modo pausa

14.4. Música y efectos sonoros

Añade una música de fondo al juego. También sería interesante añadir un efecto sonoro cuando se consigue una línea, cuando la pieza cae al pulsar la barra espaciadora o cuando se rota una pieza.

14.5. Récorde

Guardar los récorde de puntuación conseguidos. En concreto, guardar las 10 mejores puntuaciones. Cada puntuación puede venir acompañada por el nombre del usuario que la consiguió (se le pedirá al usuario que introduzca su nombre cuando consiga un nuevo record). La primera pantalla servirá para mostrar las mejores puntuaciones. El usuario podrá comenzar a jugar pulsando una tecla.

14.6. Pruebas unitarias con QUnit desde consola

Las pruebas unitarias se han realizado usando QUnit desde JSFiddle. Es posible usar QUnit directamente desde consola, mediante la ayuda de node.js, Bower o Grunt. En este ejercicio opcional el objetivo sería crear un único archivo con todas las pruebas unitarias QUnit y ejecutarlo desde consola para comprobar rápidamente si alguna de ellas falla. Como extra, sería interesante añadir una prueba unitaria más que determinara si el método `game_over` es correcto.

14.7. Refactorización del método de pintar en pantalla

Las piezas del juego actual se pintan directamente en el Canvas. Al moverse, se llama a la función que borra la posición actual y vuelve a pintar la pieza en la nueva posición. Esta forma de simular el movimiento, aunque funciona, no es fluido. Genera leves parpadeos y algunos problemas de redibujado (algunos bordes de piezas se solapan). Además, estamos forzando el redibujado de las piezas cada X milisegundos, independientemente de la carga del navegador. Para solucionar estos problemas sería interesante hacer una refactorización del método de pintar en pantalla: por un lado, se podría dibujar en un array que simulara el tablero y cuando se realizara un movimiento, volcar el array al canvas. Esto evitaría los problemas de redibujado actuales (por borrar y volver a pintar). Por otro lado, existe un método `window.requestAnimationFrame` que permite llamar a la función de redibujado en pantalla con la frecuencia máxima pero cuando el navegador considere más adecuado (ver <https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame>). De esta forma, las llamadas al pintado en pantalla no se fuerzan cada X milisegundos, sino que el navegador puede hacerlo un poco antes o un poco después, para evitar sobrecargarse de tareas.

14.8. Dos jugadores

Se trata de que dos jugadores puedan jugar a la vez con un único teclado y pantalla. Un poco incómodo si el teclado es pequeño, pero puede servir como base para la siguiente propuesta.

14.9. Juego en red

Realmente esta sería la forma idónea para dar soporte a dos jugadores. Hemos estudiado cómo trabajar con WebSockets, comunicar datos por JSON, gestionar eventos... Dispones de todo lo necesario, ¿te animas a afrontar este reto? ;-)