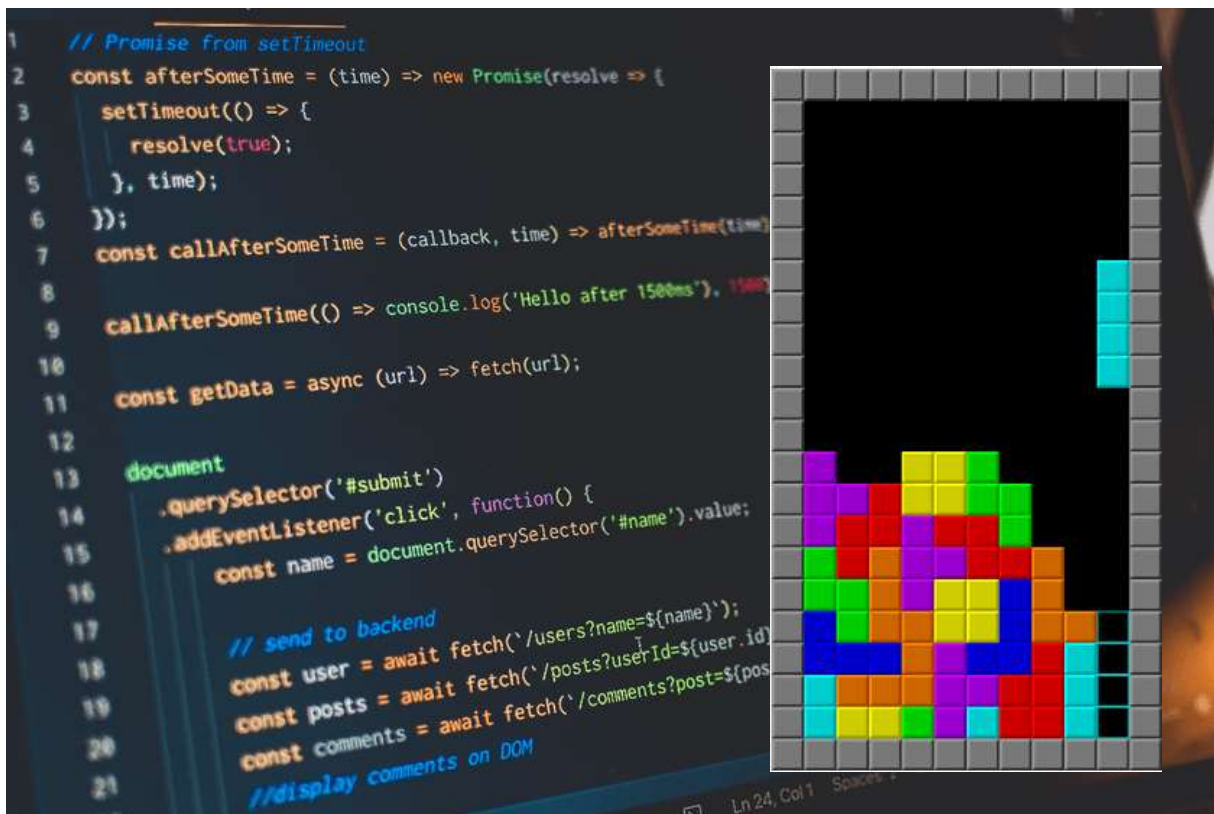


Javascriptでテトリスを作ろう！



目次

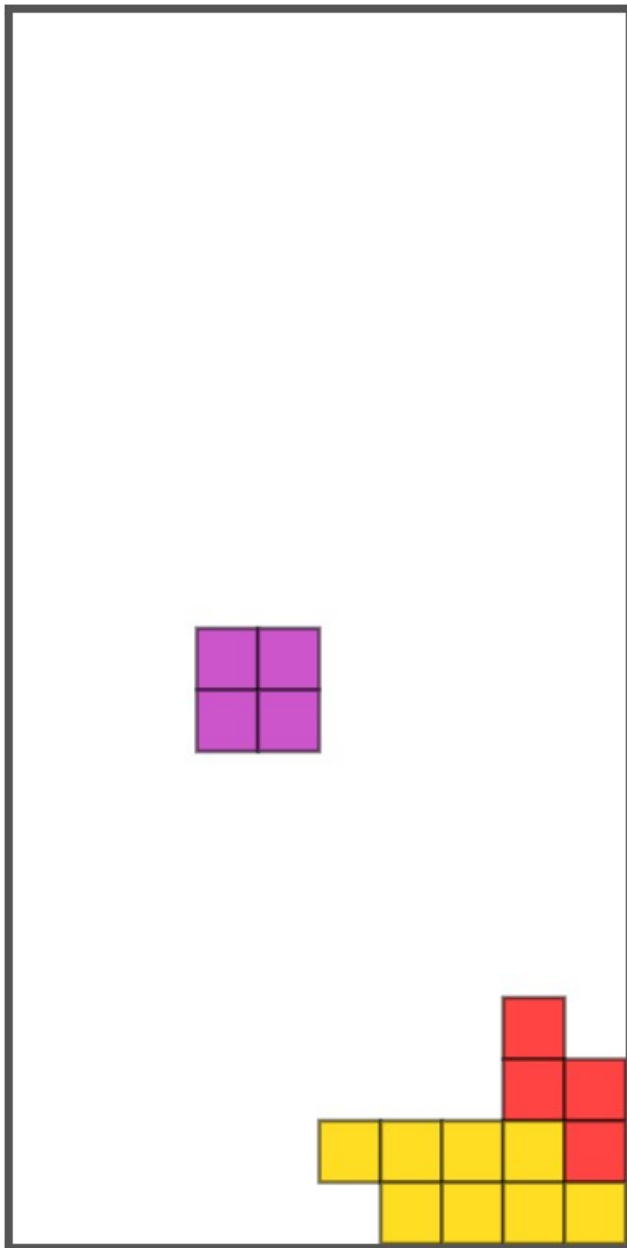
ゲームの仕様

テトリス作成に必要なAPIを覚えよう

- ・ canvasの使い方
- ・ ブロックを動かす
- ・ タイマ処理

テトリスを作ろう

今回はjavascriptを使って、テトリスを作ってみます。
完成図は以下になります。



操作方法：

ブロックの左移動…カーソルキーの【←】

ブロックの右移動…カーソルキーの【→】

強制落下…カーソルキーの【↓】。一気に一番下まで落ちます。

ブロックの回転…スペースキー

得点表示や操作方法の変更については、完成後に改造を行っていきます。

プログラミングの準備

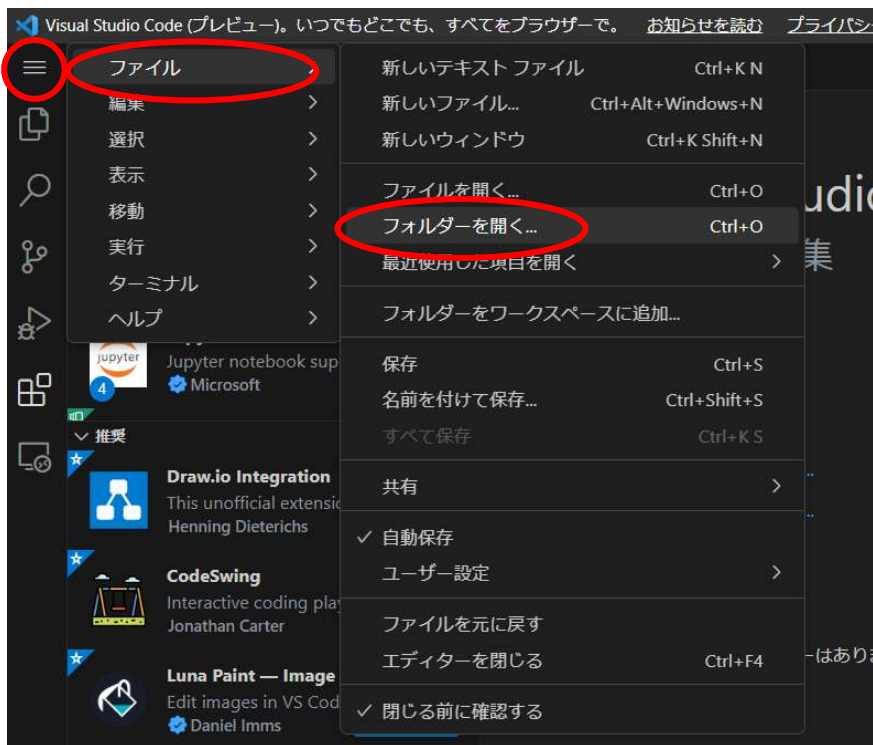
コードを書くためのエディタは、VsCodeを使用します。


<https://vscode.dev/>

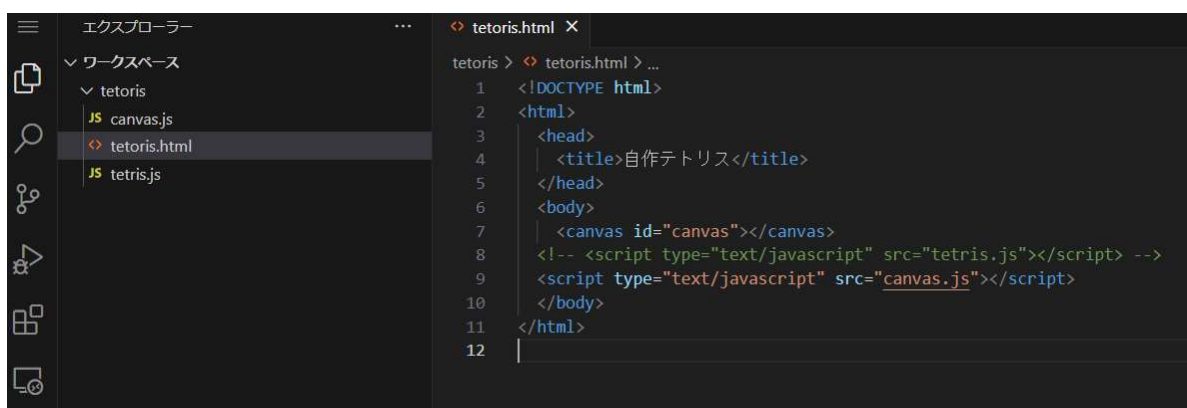
アプリはダウンロードせず、ブラウザ上で使用します。

画面左上の3本線から、ファイル→フォルダを開く…を選択し

テトリス作成用に渡した「tetoris」フォルダを選択してください



 のアイコンをクリックし、以下のように表示されれば準備完了です。



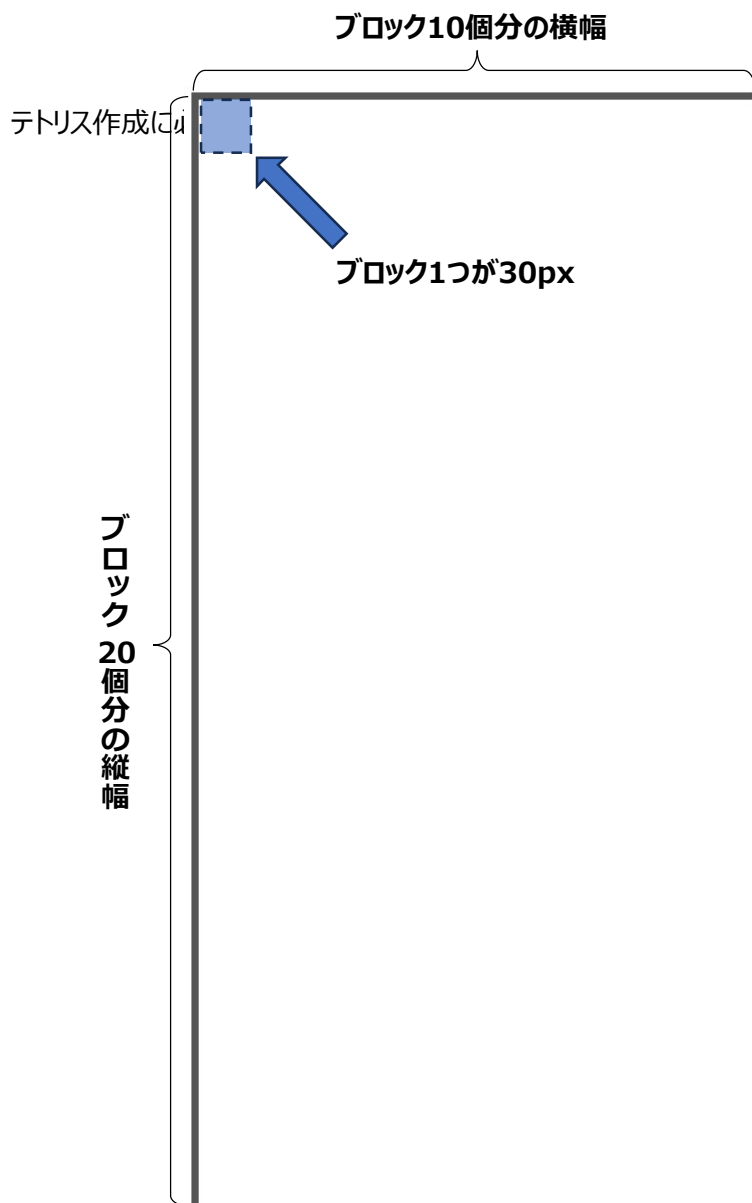
canvasを使って図形を描く

図形を表示するためにhtml/javascriptに以下を書いていきましょう。
図形を描くための領域（キャンバス）を指定する必要があります。
以下のプログラムを入力してください。

canvas.js

```
1  const BLOCK_SIZE = 30;           //ブロック1個の大きさ(30px)
2
3  const FIELD_COL = 10;             //ゲーム画面の横幅(ブロック10個分)
4  const FIELD_ROW = 20;            //ゲーム画面の縦幅(ブロック20個分)
5
6  const SCREEN_W = FIELD_COL * BLOCK_SIZE; //表示領域[横] 30×10 = 300px
7  const SCREEN_H = FIELD_ROW * BLOCK_SIZE; //表示領域[横] 30×20 = 600px
8
9
#  let can = document.getElementById("canvas"); //キャンバスの取得
#  let con = can.getContext("2d");           //2dコンテキストを取得
#  can.width = SCREEN_W;                     //キャンバスの横幅を指定
#  can.height = SCREEN_H;                   //キャンバスの縦幅を指定
#  can.style.border = "4px solid #555";     //キャンバスの枠線を指定
#
```

以下のように灰色の枠線が表示できればOKです。

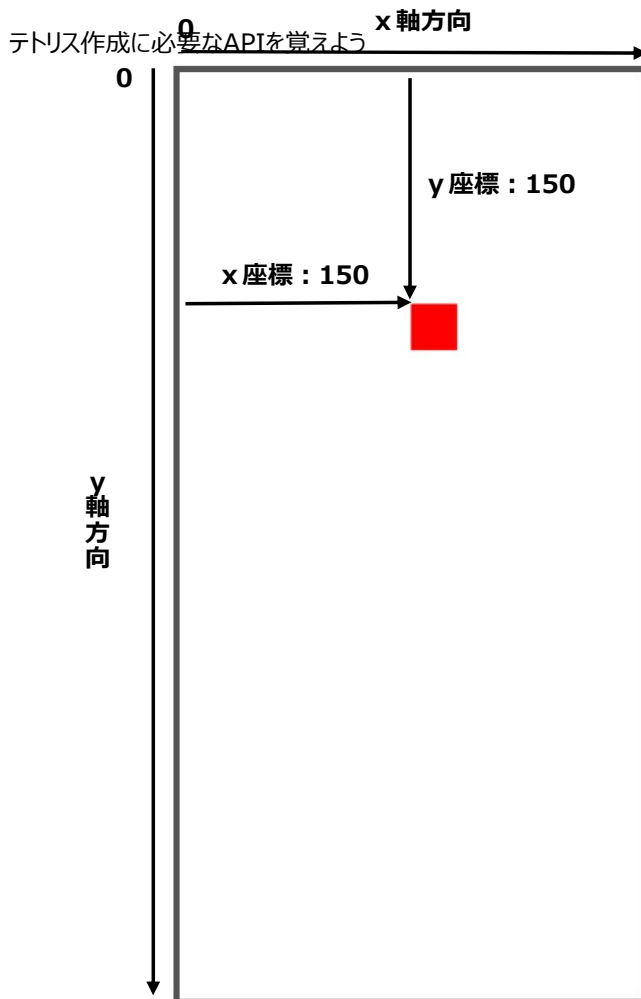


canvasを使って図形を描く（ブロックの描画）

次に、先ほど作ったキャンバスにブロック1マス分の四角を描いてみましょう。
オレンジ色の箇所プログラムを入力してください。

canvas.js

```
1  const BLOCK_SIZE = 30;           //ブロック1個の大きさ(30px)
2
3  const FIELD_COL = 10;            //ゲーム画面の横幅(ブロック10個分)
4  const FIELD_ROW = 20;           //ゲーム画面の縦幅(ブロック20個分)
5
6  const SCREEN_W = FIELD_COL * BLOCK_SIZE; //表示領域[横] 30×10 = 300px
7  const SCREEN_H = FIELD_ROW * BLOCK_SIZE; //表示領域[横] 30×20 = 600px
8
9
#  let can = document.getElementById("canvas"); //キャンバスの取得
#  let con = can.getContext("2d");           //2dコンテキストを取得
#  can.width = SCREEN_W;                     //キャンバスの横幅を指定
#  can.height = SCREEN_H;                   //キャンバスの縦幅を指定
#  can.style.border = "4px solid #555";     //キャンバスの枠線を指定
#
#  con.fillStyle="#F00";                   //ブロックの色を赤に指定
#  con.fillRect(150,150,BLOCK_SIZE,BLOCK_SIZE); //x座標150,y座標150の場所に幅30,縦30の四角を描画
```



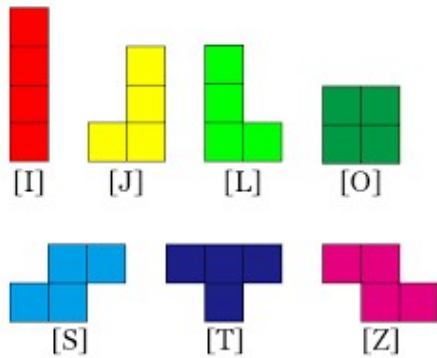
図形を表示する際、x座標/y座標の原点は
左上の角に(0,0)の座標が配置されます。

すべての要素がこの原点から配置されます。

よって赤のブロックは左から150ピクセル、上から150ピクセルの位置になります。

canvasを使って図形を描く（テトロミノの描画）

テトリスに出てくる7種類のブロックを「テトロミノ」というようです。



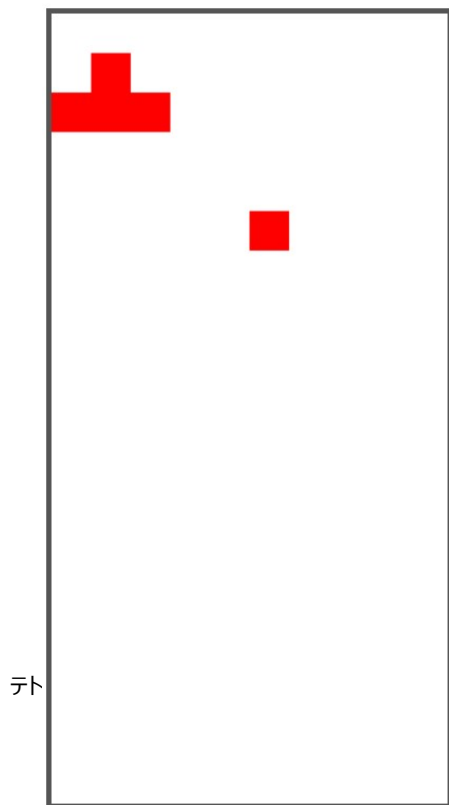
そのうちの1つをキャンバスに描画してみましょう。今回テトロミノのデータは配列を使って描画します。
オレンジ色の箇所のプログラムを入力してください。

canvas.js

```
1  const BLOCK_SIZE = 30; //ブロック1個の大きさ(30px)
2
3  const TETRO_SIZE = 4; //配列の1辺の大きさ（4×4）
4
5  const TETRO_TYPE = [ //T型のテトロミノ
6      [0,0,0,0],
7      [0,1,0,0],
8      [1,1,1,0],
9      [0,0,0,0]
10 ];
11
12 #
13 # const FIELD_COL = 10; //ゲーム画面の横幅(ブロック10個分)
14 # const FIELD_ROW = 20; //ゲーム画面の縦幅(ブロック20個分)
15 #
16 # const SCREEN_W = FIELD_COL * BLOCK_SIZE; //表示領域[横] 30×10 = 300px
17 # const SCREEN_H = FIELD_ROW * BLOCK_SIZE; //表示領域[縦] 30×20 = 600px
18 #
19 #
20 # let can = document.getElementById("canvas"); //キャンバスの取得
21 # let con = can.getContext("2d"); //2dコンテキストを取得
22 # can.width = SCREEN_W; //キャンバスの横幅を指定
23 # can.height = SCREEN_H; //キャンバスの縦幅を指定
24 # can.style.border = "4px solid #555"; //キャンバスの枠線を指定
25 #
26 # con.fillStyle="#F00"; //ブロックの色を赤に指定
27 # con.fillRect(150,150,BLOCK_SIZE,BLOCK_SIZE); //x座標150,y座標150の場所に幅30,縦30の四角を描画
28 #
29 # for (let y = 0; y < TETRO_SIZE; y++) { //テトロミノの描画
30 #     for (let x = 0; x < TETRO_SIZE; x++) {
31 #         if (TETRO_TYPE[y][x]) { //配列の中身が"1"だった場合
32 #             con.fillRect(x*BLOCK_SIZE, y*BLOCK_SIZE, //四角を描画
33 #                 BLOCK_SIZE, BLOCK_SIZE);
34 #         }
35 #     }
36 # }
37 # }
38 #
```

canvasを使って図形を描く（テトロミノの描画）

以下のように表示されれば成功です。



ポイント解説

テトロミノの1つ1つを2次元配列で表現します。
ブロックがある場合を「1」、ない場合を「0」としています。

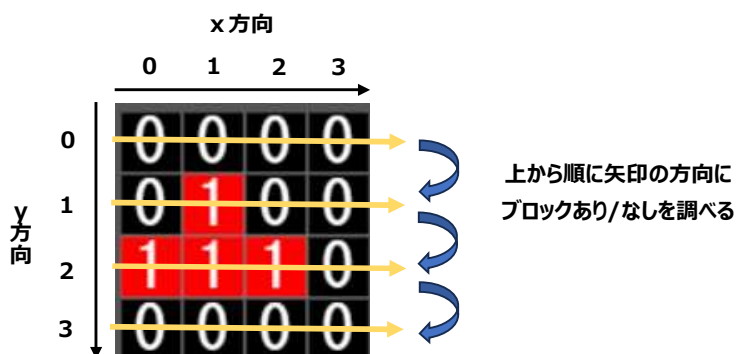
canvas.js

```
1
2  const TETRO_TYPE = [
3    [0,0,0,0],
4    [0,1,0,0],
5    [1,1,1,0],
6    [0,0,0,0]
7  ];
```

0	0	0	0
0	1	0	0
1	1	1	0
0	0	0	0

そして、キャンバスに描画するため、2重ループのfor文で上から順にブロックのあり/なしを調べていきます。

```
1
2 for (let y = 0; y < TETRO_SIZE; y++) {
3   for (let x = 0; x < TETRO_SIZE; x++) {
4     if (TETRO_TYPE[y][x]) {
5       con.fillRect(x*BLOCK_SIZE, y*BLOCK_SIZE,
6         BLOCK_SIZE, BLOCK_SIZE);
7     }
8   }
9 }
#
```



矢印キーでテトロミノを移動する

キーイベントを利用して、テトロミノを移動してみましょう。

オレンジ色の箇所のプログラムを入力してください。

canvas.js

```
1  const BLOCK_SIZE = 30; //ブロック1個の大きさ(30px)
2
3  const TETRO_SIZE = 4; //配列の1辺の大きさ (4×4)
4
5  const TETRO_TYPE = [ //T型のテトロミノ
6    [0,0,0,0],
7    [0,1,0,0],
8    [1,1,1,0],
9    [0,0,0,0]
#  ];
#
#  const FIELD_COL = 10; //ゲーム画面の横幅(ブロック10個分)
#  const FIELD_ROW = 20; //ゲーム画面の縦幅(ブロック20個分)
#
#  const SCREEN_W = FIELD_COL * BLOCK_SIZE; //表示領域[横] 30×10 = 300px
#  const SCREEN_H = FIELD_ROW * BLOCK_SIZE; //表示領域[横] 30×20 = 600px
#
#  let offsetX = 0; //テトロミノの移動量 (x方向)
#  let offsetY = 0; //テトロミノの移動量 (y方向)
#
#  let can = document.getElementById("canvas"); //キャンバスの取得
テトロ let con = can.getContext("2d"); //2dコンテキストを取得
#  can.width = SCREEN_W; //キャンバスの横幅を指定
#  can.height = SCREEN_H; //キャンバスの縦幅を指定
#  can.style.border = "4px solid #555"; //キャンバスの枠線を指定
#
#  function drawAll() { //画面描画の関数を指定
#
#    for (let y = 0; y < TETRO_SIZE; y++) { //テトロミノの描画
#      for (let x = 0; x < TETRO_SIZE; x++) {
#        if (TETRO_TYPE[y][x]) { //配列の中身が"1"だった場合
#          con.fillRect( //四角を描画
#            (offsetX+x)*BLOCK_SIZE,
#            (offsetY+y)*BLOCK_SIZE,
#            BLOCK_SIZE, BLOCK_SIZE);
#        }
#      }
#    }
#  }
#
#  document.addEventListener('keydown', //イベント処理 キーが押されたとき
#    KeyDownFunc ); //KeyDownFunc関数を呼ぶ
#
#  function KeyDownFunc(e){
#    switch (e.keyCode) { //キーコードを確認
#      case 37: //左キーを押した場合
#        offsetX--;
#        break;
#
#      case 38: //上キーを押した場合
#        offsetY--;
#        break;
#
#      case 39: //右キーを押した場合
#        offsetX++;
#        break;
#
#      case 40: //下キーを押した場合
#        offsetY++;
#        break;
#    }
#    drawAll();
#  }
#  };
```


矢印キーでテトロミノを移動する

矢印キーを押して、テトロミノが動けば成功です。

ポイント解説

① ②

```
document.addEventListener('keydown',KeyDownFunc );
```

「addEventListener」は、マウス操作やキーボード入力した際など、様々なイベント処理を行います。

- ①：イベントの種類を指定します（'keydown'は、キーボードのキーを押した時になります）。
- ②：イベントが発生したときに呼ばれる関数

1回の移動で1ブロック分移動するように変数 offsetX/offsetYを用意しました。
キー操作をすることで、テトロミノを移動できます。

キーの種類とキーコードの対応表を以下に示します。
赤枠で囲った箇所が矢印キーのキーコードになります。

テ

アルファベット				数字		テンキー数字		テンキー記号		記号	
A	65	O	79	0	48	T0	96	T*	106	:*	186
B	66	P	80	1	49	T1	97	T+	107	:+	187
C	67	Q	81	2	50	T2	98			,<	188
D	68	R	82	3	51	T3	99	T-	109	-=	189
E	69	S	83	4	52	T4	100	T.	110	.>	190
F	70	T	84	5	53	T5	101	T/	111	/?	191
G	71	U	85	6	54	T6	102			@`	192
H	72	V	86	7	55	T7	103			[[219
I	73	W	87	8	56	T8	104			¥	220
J	74	X	88	9	57	T9	105]]	221
K	75	Y	89							^^	222
L	76	Z	90							¥_	226
M	77										
N	78										

ファンクションキー		制御キー			
F1 (ヘルプ)	112	BackSpace	8	End	35 英数 240
F2	113	NumLockOFFのT5	12	Home	36 カタカナ/ひらがな 242
F3 (検索)	114	Enter / T Enter	13	←	37 Esc 243
F4 (アドレスバー)	115	Shift	16	↑	38 半角/全角 244
F5 (更新)	116	Ctrl	17	→	39 Tab 9
F6 (フォーカス)	117	Alt	18	↓	40
F7	118	Pause	19	Insert	45
F8	119	変換	28	Delete	46
F9	120	無変換	29	Win	91
F10 (Alt)	121	スペース	32	Apps	93
F11 (全画面)	122	PageUp	33	NumLock	144
F12	123	PageDown	34	ScrollLock	145

タイマ処理でブロックを動かす

次にテトロミノを一定間隔で下に移動させてみます。
オレンジ色の箇所のプログラムを入力してください。

canvas.js

```
1  const BLOCK_SIZE = 30; //ブロック1個の大きさ(30px)
2
3  const TETRO_SIZE = 4; //配列の1辺の大きさ (4×4)
4
5  const TETRO_TYPE = [ //T型のテトロミノ
6    [0,0,0,0],
7    [0,1,0,0],
8    [1,1,1,0],
9    [0,0,0,0]
10 ];
11
12 #
13 #
14 # const GAME_SPEED = 1000; //ゲームスピード
15 #
16 # const FIELD_COL = 10; //ゲーム画面の横幅(ブロック10個分)
17 # const FIELD_ROW = 20; //ゲーム画面の縦幅(ブロック20個分)
18 #
19 # const SCREEN_W = FIELD_COL * BLOCK_SIZE; //表示領域[横] 30×10 = 300px
20 # const SCREEN_H = FIELD_ROW * BLOCK_SIZE; //表示領域[横] 30×20 = 600px
21 #
22 # let offsetX = 0; //テトロミノの移動量 (x方向)
23 # let offsetY = 0; //テトロミノの移動量 (y方向)
24
25 テトリス作成に必要なAPIを覚えよう
26 # let can = document.getElementById("canvas"); //キャンバスの取得
27 # let con = can.getContext("2d"); //2dコンテキストを取得
28 # can.width = SCREEN_W; //キャンバスの横幅を指定
29 # can.height = SCREEN_H; //キャンバスの縦幅を指定
30 # can.style.border = "4px solid #555"; //キャンバスの枠線を指定
31 #
32 # let timerID = setInterval(droptetro, GAME_SPEED); //インターバル処理
33 # //一定間隔でdroptetro関数を呼び出す
34 #
35 # function drawAll() { //画面描画の関数
36 #
37 #   for (let y = 0; y < TETRO_SIZE; y++) { //テトロミノの描画
38 #     for (let x = 0; x < TETRO_SIZE; x++) {
39 #       if (TETRO_TYPE[y][x]) { //配列の中身が"1"だった場合
40 #         con.fillRect( //四角を描画
41 #           (offsetX+x)*BLOCK_SIZE,
42 #           (offsetY+y)*BLOCK_SIZE,
43 #           BLOCK_SIZE, BLOCK_SIZE);
44 #       }
45 #     }
46 #   }
47 # }
48 #
49 # function droptetro() { //テトロミノを一定間隔で下に移動
50 #   offsetY++; //ブロックのY座標を1ブロック分下に指定
51 #   drawAll(); //画面描画関数を呼ぶ
52 # }
53 #
54 # document.addEventListener('keydown', //イベント処理 キーが押されたとき
55 #                           KeyDownFunc ); //KeyDownFunc関数を呼ぶ
56 #
57 # function KeyDownFunc(e){
58 #   switch (e.keyCode) {
59 #     case 37: //キーコードを確認
60 #       //左キーを押した場合
61 #       offsetX--;
62 #       break;
63 #     case 38: //上キーを押した場合
64 #       offsetY--;
65 #       break;
66 #     case 39: //右キーを押した場合
67 #       offsetX++;
68 #       break;
69 #     case 40: //下キーを押した場合
70 #       offsetY++;
71 #       break;
72 #   }
73 #   drawAll();
74 # };
```

タイマ処理でブロックを動かす

0.5秒ごとにテトロミノが下に移動すれば成功です。

ポイント解説

①

②

setInterval(droptetro, GAME_SPEED);

「setInterval」は、一定間隔で関数を呼び出すことができます。

①：イベントが発生したときに呼ばれる関数

②：タイマ値（単位はミリ秒）。例えば1000と指定した場合は1秒になります。

テトリス作成に必要なAPIを覚えよう

テトリスを作ろう

tetoris.htmlより、8行目のコメントを外し、9行目をコメント化してください。

```
6 <body>
7   <canvas id="canvas"></canvas>
8   <script type="text/javascript" src="tetris.js"></script>
9   <!-- <script type="text/javascript" src="canvas.js"></script> -->
10 </body>
```

以下がテトリスのプログラムになります。

tetoris.jsに入力していきましょう。

tetoris.js

```
1  const BLOCK_SIZE = 30; //ブロック1個の大きさ(30px)
2
3  const TETRO_SIZE = 4; //配列の1辺の大きさ (4×4)
4
5  const FIELD_COL = 10; //幅 //ゲーム画面の横幅(ブロック10個分)
6  const FIELD_ROW = 20; //高さ //ゲーム画面の縦幅(ブロック20個分)
7
8  const SCREEN_W = FIELD_COL * BLOCK_SIZE; //表示領域[横] 30×10 = 300px
9  const SCREEN_H = FIELD_ROW * BLOCK_SIZE; //表示領域[横] 30×20 = 600px
10
11  const TETRO_COLOR = [ //テトロミノの色
12    "#000", //0:空
13    "#6CF", //1:水色
14    "#F92", //2:オレンジ
15    "#66F", //3:青
16    "#C5C", //4:紫
17    "#FD2", //5:黄色
18    "#F44", //6:赤
19    "#5B5", //7:緑
20  ];
21
22  const TETRO_TYPE = [ //テトロミノの型
23    [], // 0:空データ
24    [ // 1:I型
25      [0,0,0,0],
26      [1,1,1,1],
27      [0,0,0,0],
28      [0,0,0,0]
29    ],
30    [ // 2:L型
31      [0,1,0,0],
32      [0,1,0,0],
33      [0,1,1,0],
34      [0,0,0,0]
35    ],
```

```

36  [ // 3:I型
37  [0,0,1,0],
38  [0,0,1,0],
39  [0,1,1,0],
40  [0,0,0,0]
41  ],
42  [ // 4:O型
43  [0,0,0,0],
44  [0,1,1,0],
45  [0,1,1,0],
46  [0,0,0,0]
47  ],
48  [ // 5:Z型
49  [0,0,0,0],
50  [1,1,0,0],
51  [0,1,1,0],
52  [0,0,0,0]
53  ],
54  [ // 6:S型
55  [0,0,0,0],
56  [0,1,1,0],
57  [1,1,0,0],
58  [0,0,0,0]
59  ],
60  [ // 7:T型
61  [0,1,0,0],
62  [0,1,1,0],
63  [0,1,0,0],
64  [0,0,0,0]
65  ],
66 ];
67
68 const GAME_SPEED = 800; //ゲームスピード（0.8秒）
69
70 let tetro = []; // 変数（テトロミノ配列）
71 let tetro_x = 0; // 変数（テトロミノ X座標）
72 let tetro_y = 0; // 変数（テトロミノ Y座標）
73 let field = []; // 変数（ゲーム画面配列）
74 let tetro_type; // 変数（テトロミノの型）
75 let game_over = false; // 変数（ゲームオーバーフラグ）
76
77 let can = document.getElementById("canvas"); //キャンバスの取得
78 let con = can.getContext("2d"); //2dコンテキストを取得
79 can.width = SCREEN_W; //キャンバスの横幅を指定
80 can.height = SCREEN_H; //キャンバスの縦幅を指定
81 can.style.border = "4px solid #555"; //キャンバスの枠線を指定
82
83 let timerID = setInterval(droptetro, GAME_SPEED); //インターバルタイマの設定
84 //0.8秒毎にdroptetro関数を呼ぶ
85 init(); //ゲーム初期化
86 drawAll(); //画面描画
87

```

```

// 関数：フィールドの初期化
88 function init(){
89   for(let y=0; y<FIELD_ROW; y++)
90   {
91     field[y] = [];
92     for(let x=0; x<FIELD_COL; x++)
93     {
94       field[y][x] = 0; //配列fieldを全て「0」
95     } //ブロック無しに設定する
96   }
97   tetrominoSelect(); //ゲーム開始時のテトロミノを指定
98 }
99
// 関数：テトロミノをランダムに選択
100 function tetrominoSelect(){
101   tetromino_type = Math.floor(Math.random() //テトロミノをランダムで選択
102     *(TETROMINO_TYPE.length-1)) + 1;
103   tetromino = TETROMINO_TYPE[tetromino_type];
104 }
105
// 関数：ブロックを描画する
106 function drawBlock(x, y, color) {
107   let px = x * BLOCK_SIZE; //ブロック表示位置 (x座標)
108   let py = y * BLOCK_SIZE; //ブロック表示位置 (y座標)
109   con.fillStyle=TETROMINO_COLOR[color]; //ブロックの色を指定
110   con.fillRect(px,py,BLOCK_SIZE,BLOCK_SIZE); //四角を描画
111   con.strokeStyle = "black"; //ブロックの枠線を黒にする
112   con.strokeRect(px,py,BLOCK_SIZE,BLOCK_SIZE); //ブロックの枠線を描画
113 }
114
// 関数：画面全体を描画する
115 function drawAll(){
116   con.clearRect(0,0,SCREEN_W,SCREEN_H); //画面全体を一旦クリアする
117
118   for(let y=0; y<FIELD_ROW; y++)
119   {
120     for(let x=0; x<FIELD_COL; x++)
121     {
122       if(field[y][x])
123       {
124         drawBlock(x, y, field[y][x]);
125       }
126     }
127   }
128
129   for(let y=0; y<TETROMINO_SIZE; y++)
130   {
131     for(let x=0; x<TETROMINO_SIZE; x++)
132     {
133       if(tetromino[y][x])
134       {
135         drawBlock(tetromino_x + x, tetromino_y + y, tetromino_type);
136       }
137     }
138   }

```

落下済みのブロックを表示

落下中のブロックを表示

```

139
140 if(game_over){ //ゲームオーバーになったら
141     let str = "GAME OVER";
142     con.font = "40px 'M S ゴシック'";
143     let w = con.measureText(str).width;
144     let x = SCREEN_W/2 - w/2;
145     let y = SCREEN_H/2;
146     con.lineWidth = 4;
147     con.strokeText(str,x,y);
148     con.fillStyle = "White";
149     con.fillText(str,x,y);
150
151     clearInterval(timerID); //インターバルタイマを止める
152 }
153 }
154
// 関数 : テトロミノが移動可能かを判定する
155 function checkMove(mx, my, new_tetro){
156
157     if(new_tetro == null) new_tetro = tetro //引数「new_tetro」が指定無しの場合
158                                           //変数「tetro」を使う
159     for(let y=0; y<TETRO_SIZE; y++)
160     {
161         for(let x=0; x<TETRO_SIZE; x++)
162         {
163             let nx = mx + tetro_x + x;
164             let ny = my + tetro_y + y;
165             if(new_tetro[y][x])
166             {
167                 if(nx < 0 //移動する先のx,y座標が
168                     || ny < 0 //画面外の場合、または
169                     || nx >= FIELD_COL //すでにブロックが存在する場合
170                     || ny >= FIELD_ROW
171                     || field[ny][nx]){
172                 return false; //移動不可と判定
173             }
174         }
175     }
176 } //上の判定以外の場合は
177 return true; //移動可能と判定
178 }
179
// 関数 : テトロミノを回転させる
180 function rotate(){
181
182     let new_tetro = [];
183     for(let y=0; y<TETRO_SIZE; y++)
184     {
185         new_tetro[y] = [];
186         for(let x=0; x<TETRO_SIZE; x++)
187         {
188             new_tetro[y][x] = tetro[TETRO_SIZE-x-1][y];
189         }
190     }
191     return new_tetro;
192 }

```

画面中央に
「GAME OVER」と表示

現在のテトロミノを
右回転させる

```

193 // 関数 : テトロミノを固定する
194 function fixtetro(){
195     for(let y=0; y<TETRO_SIZE; y++)
196     {
197         for(let x=0; x<TETRO_SIZE; x++)
198         {
199             if(tetro[y][x])
200             {
202                 field[tetro_y + y][tetro_x + x] = tetro_type; //ブロックの形（色）で固定する
203             }
204         }
205     }
206 }
207
208 // 関数 : ラインを消す
209 function clearline(){
210     for(let y=0; y<FIELD_ROW; y++) //画面の一番上から検索する
211     {
212         let line_flg = true;
213         for(let x=0; x<FIELD_COL; x++) //x座標の検索
214         {
215             if(!field[y][x]) //x方向に1つでも空があった場合
216             {
217                 line_flg = false; //そのラインは消せないと判断
218                 break; //次のラインの検索を行う
219             }
220         }
221         if(line_flg) { //ラインを消す場合
222             for(let ny = y; ny>0; ny--){
223                 for(let nx = 0; nx<FIELD_COL; nx++){
224                     field[ny][nx] = field[ny-1][nx]; //消す対象のラインに
225                 } //その上のラインを上書きする
226             }
227         }
228     }
229 }
230
231 // 関数 : テトリスを落下させる
232 function droptetro(){
233     if(checkMove(0,1)){ //落下中のテトロミノが下に移動可能か
234         tetro_y++; //y座標に+1する
235     } else{ //落下中のテトロミノが下に移動不可
236         fixtetro(); //テトロミノを固定する
237         clearline(); //ラインを消す
238         tetroTypeSelect(); //次のテトロミノを選択する
239         tetro_x = 0; //テトロミノのx方向を初期値に戻す
240         tetro_y = 0; //テトロミノのy方向を初期値に戻す
241     }
242     if(!checkMove(0,0)){ //テトロミノが移動できない
243         game_over = true; //ブロックが上まで積みあがった場合
244     } //ゲームオーバーとする
245     drawAll(); //画面全体を描画する
246 }

```



```
247 document.addEventListener('keydown', //キーイベントを設定  
                                KeyDownFunc );  
// 関数：キーが押されたときに呼ばれる関数  
249 function KeyDownFunc(e){  
250  
251     if(game_over) return;          //ゲームオーバーとなった場合  
252                                     //以降の処理は行わない  
  
253     var key_code = e.keyCode;  
254  
255     switch(key_code){  
256         case 37: //左に移動           //左キー「←」を押した  
257             if(checkMove(-1,0))tetra_x--;  
258             break;                   //テトロミノが左に移動可能か  
  
259         case 39: //右に移動           //左キー「→」を押した  
260             if(checkMove(1,0))tetra_x++;  
261             break;                  //テトロミノが右に移動可能か  
  
262         case 40: //下に移動           //左キー「↓」を押した  
263             while(checkMove(0,1))tetra_y++;  
264             break;                 //テトロミノが下に移動可能か  
  
265         case 32:                     //スペースキーを押した  
266             tetra = rotate();        //テトロミノを回転  
267             if(checkMove(0,0, tetra)) tetra = tetra;  
268             break;  
269     }  
270 drawAll();                          //画面全体を描画する  
271 }
```