



第17回
日本e-learning大賞
「日本電子出版協会会長賞」

SEGA CHALLENGE!

ふよふよプログラミング

◆◆JavaScript & HTML5◆◆

サンプルコード

『ぶよぶよプログラミング』著作物利用許諾書

はじめに

お客様が『ぶよぶよプログラミング』で使用している著作物（以下「本プログラム等」といいます）を利用するにあたり、必ず下記記載の各内容（以下「本利用条件」といいます）をご覧ください。

お客様が本利用条件にご同意いただけることを条件に、株式会社セガ（以下「当社」といいます）は、お客様に対して本プログラム等の利用を許諾いたします。

お客様が、本プログラム等の利用を開始したことをもって、お客様が本利用条件に同意し、この条件に従って本プログラム等を利用することに同意したものとみなされます。

もし、ソフトウェア使用許諾契約書にご同意いただけない場合には、本プログラム等のご利用をお止めいただき、直ちに、本プログラム等の掲載サイトの接続を中止してください。

記

1. 趣旨

当社はお客様（以下「使用者」といいます）に対し、使用者が本プログラム等を以下の使用条件で使用する権利を許諾します。本利用条件は、使用者が本プログラム等のご利用を開始したときに、その効力を生じます。

プログラミング基礎学習のために利用すること（以下「本利用目的」といいます）。

2. 著作権

本プログラム等に関する著作権その他の権利は当社が保有していることを確認します。使用者は本利用条件によって許諾されている以外、本プログラム等に関するいかなる権利をも取得することはできません。なお、当社は、本利用条件により、第三者または当社の本プログラム等に関連する特許等の産業財産権について、これを実施許諾するものではありません。

本利用目的の範囲内において、本利用条件に基づき改変した本プログラム等の著作権（著作権法第21条から第28条までに定めるすべての権利を意味します）は、当社に無償にて譲渡及び移転するものとします。また、使用者は、当社が当該本プログラム等を、当社の判断で、商用であるか否かにかかわらず、いかなる方法にて、利用、公開、公表、販売もしくは頒布等を行うことができることを承諾するものとします。

3. 使用許諾

当社は、使用者に対し次の各号に定める権利を許諾します。

- ・本利用目的の範囲内において、本プログラム等を改変する権利
- ・本利用目的の範囲内において、本プログラム等に含まれている画像の拡大及び縮小をする権利
- ・本プログラム等（前号により改変された本プログラム等も含む。以下同じ。）を、非商用目的においてのみ、第三者に公開する権利

4. 禁止事項

使用者は本プログラム等の使用にあたり、次に記載される行為をすることができません。

- ・本利用目的またはその他本利用条件で許諾されていない方法で本プログラム等を使用すること。
- ・当社が使用者に開示した本プログラム等のソースコードを、本利用条件を承諾していない第三者に開示し、当該第三者の利用に供すること及びそのおそれがある行為をすること。
- ・方法の如何を問わず、本プログラム等を商用利用のために利用すること。なお、商用利用をご希望される場合は、当社に書面または電子メールにてご連絡いただき、当社と別途商用利用のための契約書を締結いただくことによるのみ、商用利用することができるものとします。但し、使用者からのご連絡に基づき、当該契約書を締結するか否かは、当社の判断によるものとします。
- ・本プログラム等における著作権表示を削除し、または改変すること。
- ・本利用条件に定められた内容を除き、本プログラム等における当社の著作権及び産業財産権を侵害する行為または第三者の権利の侵害を行うこと。
- ・上記記載の各内容は、本利用条件に基づき、使用者が改変した本プログラム等に対しても適用されるものとします。

使用者が、上記の禁止事項のいずれかに該当し、当社または第三者に何らかの損害が生じた場合、当社及び当該第三者に対してその損害の賠償を行う責任を負うものとします。

5. 保証

当社は、本プログラム等についてバグを含む論理上の誤りが無いことまたは正しく動作することを含む機能上の正確性、本プログラム等におけるいかなる欠陥もないこと、第三者の権利を侵害しないこと等を含み、いかなる保証も行わないものとします。

万が一、当社により、本プログラム等に何等かのバグを含む論理上の誤り、正しく動作しないことを含む機能上の問題、その他本プログラム等におけるいかなる欠陥等が発見された場合であっても、当社はそれらを修正、改修等を行う責任を負うものではありません。

本利用条件に基づく使用者による本プログラム等の使用から生じた、使用者と第三者との間のトラブル・紛争等について、当社はいかなる責任を負うものではありません。

6. 契約の終了

本利用規約は、以下のいずれかの時点まで有効とします。

- ・当社が、本利用規約に基づく本プログラム等の利用の中止を判断した場合。
- ・本プログラム等を掲載しているサービスが終了した場合。
- ・使用者が本利用条件の条項のいずれかに違反した場合には、当社から通告することなく、直ちに本利用条件は解除されます。なお、この場合において、当社または第三者に何らかの損害が生じた場合、当社及び当該第三者に対してその損害の賠償を行う責任を負うものとします。
- ・前述に基づく、当社による本プログラム等の利用の中止、または使用者による本利用条件の条項のいずれかに違反したことによる本利用条件の解除のいずれかがなされた場合、使用者は、本プログラム等をアンインストールし、本プログラム等及びそれらの複製物すべてを直ちに破棄しなければならないものとします。

7. 譲渡禁止

使用者は、当社の書面による承諾を得ない限り、本プログラム等及び本利用条件に基づく権利または義務を第三者に譲渡することはできません。

8. 準拠法及び分離性

日本法を準拠法として、同法によって解釈されるものです。本利用条件の中のある条項が裁判所によって無効と判断された場合でも、残りの条項は効力を有します。

9. 契約の完全合意性

本利用条件は、本プログラム等の使用について、使用者と当社の間で取り決められた内容のすべてを記載するものであり、本件に関して、今までに取り交わした契約（口頭、文書の両方を含みます）に優先して適用されるものです。

以上

株式会社セガ
2020年6月25日

サンプルコード

index.html

```
1 <!DOCTYPE HTML>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
7   <meta http-equiv="Content-Security-Policy"
8     content="default-src * data: gap: content: https://ssl.gstatic.com; style-src * 'unsafe-inline'; script-src *
9     'unsafe-inline' 'unsafe-eval'">
10  <title>ぶよぶよの制作を体験してみよう！</title>
11  <link rel="stylesheet" href="components/loader.css">
12  <link rel="stylesheet" href="css/style.css">
13 </head>
14 <body style="margin:0;">
15   <!-- <div id="stage" style="position:absolute; left: 0; top: 0; overflow: hidden;"></div> -->
16   <div id="stage" style="position:relative; margin: 0 auto; overflow: hidden;background: url(img/puyo_4bg.png)"></div>
17   <!-- <div id="score" style="position:absolute; left: 0; top: 0; overflow: hidden; text-align: right;"></div> -->
18   <div id="score" style="margin: 0 auto; overflow: hidden; text-align: right;"></div>
19   <div style="display:none">
20     
21     
22     
23     
24     
25     
26     
27     
28     
29     
30     
31     
32     
33     
34     
35     
36     
37   </div>
38   <script src="components/loader.js"></script>
39   <script src="src/config.js"></script>
40   <script src="src/stage.js"></script>
41   <script src="src/score.js"></script>
42   <script src="src/puyoimage.js"></script>
43   <script src="src/player.js"></script>
44   <script src="src/game.js"></script>
45 </body>
46
47 </html>
```

game.js

```
1 // 起動されたときに呼ばれる関数を登録する
2 window.addEventListener("load", () => {
3     // まずステージを整える
4     initialize();
5
6     // ゲームを開始する
7     loop();
8 });
9
10 let mode; // ゲームの現在の状況
11 let frame; // ゲームの現在フレーム (1/60秒ごとに1追加される)
12 let combinationCount = 0; // 何連鎖かどうか
13
14 function initialize() {
15     // 画像を準備する
16     PuyoImage.initialize();
17     // ステージを準備する
18     Stage.initialize();
19     // ユーザー操作の準備をする
20     Player.initialize();
21     // シーンを初期状態にセットする
22     Score.initialize();
23     // スコア表示の準備をする
24     mode = 'start';
25     // フレームを初期化する
26     frame = 0;
27 }
28
29 function loop() {
30     switch(mode) {
31         case 'start':
32             // 最初は、もしかしたら空中にあるかもしれないぷよを自由落下させるところからスタート
33             mode = 'checkFall';
34             break;
35         case 'checkFall':
36             // 落ちるかどうか判定する
37             if(Stage.checkFall()) {
38                 mode = 'fall'
39             } else {
40                 // 落ちないならば、ぷよを消せるかどうか判定する
41                 mode = 'checkErase';
42             }
43             break;
44         case 'fall':
45             if(!Stage.fall()) {
46                 // すべて落ちきったら、ぷよを消せるかどうか判定する
47                 mode = 'checkErase';
48             }
49             break;
50         case 'checkErase':
```

```
51 // 消せるかどうか判定する
52 const eraseInfo = Stage.checkErase(frame);
53 if(eraseInfo) {
54     mode = 'erasing';
55     combinationCount++;
56     // 得点を計算する
57     Score.calculateScore(combinationCount, eraseInfo.piece, eraseInfo.color);
58     Stage.hideZenkeshi();
59 } else {
60     if(Stage.puyoCount === 0 && combinationCount > 0) {
61         // 全消しの処理をする
62         Stage.showZenkeshi();
63         Score.addScore(3600);
64     }
65     combinationCount = 0;
66     // 消せなかったら、新しいぷよを登場させる
67     mode = 'newPuyo'
68 }
69 break;
70 case 'erasing':
71     if(!Stage.erasing(frame)) {
72         // 消し終わったら、再度落ちるかどうか判定する
73         mode = 'checkFall';
74     }
75     break;
76 case 'newPuyo':
77     if(!Player.createNewPuyo()) {
78         // 新しい操作ぷよを作成出来なかったら、ゲームオーバー
79         mode = 'gameOver';
80     } else {
81         // プレイヤーが操作可能
82         mode = 'playing';
83     }
84     break;
85 case 'playing':
86     // プレイヤーが操作する
87     const action = Player.playing(frame);
88     mode = action; // 'playing' 'moving' 'rotating' 'fix' のどれかが帰ってくる
89     break;
90 case 'moving':
91     if(!Player.moving(frame)) {
92         // 移動が終わったので操作可能にする
93         mode = 'playing';
94     }
95     break;
96 case 'rotating':
97     if(!Player.rotating(frame)) {
98         // 回転が終わったので操作可能にする
99         mode = 'playing';
100 }
```

101	break;
102	case 'fix':
103	// 現在の位置でぷよを固定する
104	Player.fix();
105	// 固定したら、まず自由落下を確認する
106	mode = 'checkFall'
107	break;
108	case 'gameOver':
109	// ばたんきゅーの準備をする
110	PuyoImage.prepareBatankyu(frame);
111	mode = 'batankyu';
112	break;
113	case 'batankyu':
114	PuyoImage.batankyu(frame);
115	Player.batankyu();
116	break;
117	}
118	frame++;
119	requestAnimationFrame(loop); // 1/60秒後にもう一度呼び出す
120	}
121	

puyoimage.js

```
1 class PuyoImage {
2
3     // static puyoImages;
4     // static batankyuImage;
5     // static gameOverFrame;
6
7     static initialize() {
8         this.puyoImages = [];
9         for(let i = 0; i < 5; i++) {
10            const image = document.getElementById(`puyo_${i + 1}`);
11            image.removeAttribute('id');
12            image.width = Config.puyoImgWidth;
13            image.height = Config.puyoImgHeight;
14            image.style.position = 'absolute';
15            this.puyoImages[i] = image;
16        }
17        this.batankyuImage = document.getElementById('batankyu');
18        this.batankyuImage.width = Config.puyoImgWidth * 6;
19        this.batankyuImage.style.position = 'absolute';
20    }
21
22    static getPuyo(index) {
23        const image = this.puyoImages[index - 1].cloneNode(true);
24        return image;
25    }
26
27    static prepareBatankyu(frame) {
28        this.gameOverFrame = frame;
29        Stage.stageElement.appendChild(this.batankyuImage);
30        this.batankyuImage.style.top = -this.batankyuImage.height + 'px';
31    }
32
33    static batankyu(frame) {
34        const ratio = (frame - this.gameOverFrame) / Config.gameOverFrame;
35        const x = Math.cos(Math.PI / 2 + ratio * Math.PI * 2 * 10) * Config.puyoImgWidth;
36        const y = Math.cos(Math.PI + ratio * Math.PI * 2) * Config.puyoImgHeight * Config.stageRows / 4 + Config.puyoImgHeight * Config.stageRows / 2;
37        this.batankyuImage.style.left = x + 'px';
38        this.batankyuImage.style.top = y + 'px';
39    }
40 }
41
```

player.js

```
1 class Player {
2   // static centerPuyo;
3   // static movablePuyo;
4   // static puyoStatus;
5   // static centerPuyoElement;
6   // static movablePuyoElement;
7
8   // static groundFrame;
9   // static keyStatus;
10
11  // static actionStartFrame;
12  // static moveSource;
13  // static moveDestination;
14  // static rotateBeforeLeft;
15  // static rotateAfterLeft;
16  // static rotateFromRotation;
17
18  static initialize () {
19    // キーボードの入力を確認する
20    this.keyStatus = {
21      right: false,
22      left: false,
23      up: false,
24      down: false
25    };
26    // ブラウザのキーボードの入力を取得するイベントリスナを登録する
27    document.addEventListener('keydown', (e) => {
28      // キーボードが押された場合
29      switch(e.keyCode) {
30        case 37: // 左向きキー
31          this.keyStatus.left = true;
32          e.preventDefault(); return false;
33        case 38: // 上向きキー
34          this.keyStatus.up = true;
35          e.preventDefault(); return false;
36        case 39: // 右向きキー
37          this.keyStatus.right = true;
38          e.preventDefault(); return false;
39        case 40: // 下向きキー
40          this.keyStatus.down = true;
41          e.preventDefault(); return false;
42      }
43    });
44    document.addEventListener('keyup', (e) => {
45      // キーボードが離された場合
46      switch(e.keyCode) {
47        case 37: // 左向きキー
48          this.keyStatus.left = false;
49          e.preventDefault(); return false;
50        case 38: // 上向きキー
```



```

51         this.keyStatus.up = false;
52         e.preventDefault(); return false;
53         case 39: // 右向きキー
54             this.keyStatus.right = false;
55             e.preventDefault(); return false;
56         case 40: // 下向きキー
57             this.keyStatus.down = false;
58             e.preventDefault(); return false;
59     }
60 });
61 // タッチ操作追加
62 this.touchPoint = {
63     xs: 0,
64     ys: 0,
65     xe: 0,
66     ye: 0
67 }
68 document.addEventListener('touchstart', (e) => {
69     this.touchPoint.xs = e.touches[0].clientX
70     this.touchPoint.ys = e.touches[0].clientY
71 })
72 document.addEventListener('touchmove', (e) => {
73     // 指が少し動いた時は無視
74     if (Math.abs(e.touches[0].clientX - this.touchPoint.xs) < 20 &&
75         Math.abs(e.touches[0].clientY - this.touchPoint.ys) < 20
76     ) {
77         return
78     }
79
80     // 指の動きをからジェスチャーによるkeyStatusプロパティを更新
81     this.touchPoint.xe = e.touches[0].clientX
82     this.touchPoint.ye = e.touches[0].clientY
83     const {xs, ys, xe, ye} = this.touchPoint
84     gesture(xs, ys, xe, ye)
85
86
87     this.touchPoint.xs = this.touchPoint.xe
88     this.touchPoint.ys = this.touchPoint.ye
89 })
90 document.addEventListener('touchend', (e) => {
91     this.keyStatus.up = false
92     this.keyStatus.down = false
93     this.keyStatus.left = false
94     this.keyStatus.right = false
95 })
96
97 // ジェスチャーを判定して、keyStatusプロパティを更新する関数
98 const gesture = (xs, ys, xe, ye) => {
99     const horizonDirection = xe - xs;
100     const verticalDirection = ye - ys;

```

```

101
102     if (Math.abs(horizonDirection) < Math.abs(verticalDirection)) {
103         // 縦方向
104         if (verticalDirection < 0) {
105             // up
106             this.keyStatus.up = true
107             this.keyStatus.down = false
108             this.keyStatus.left = false
109             this.keyStatus.right = false
110         } else if (0 <= verticalDirection) {
111             // down
112             this.keyStatus.up = false
113             this.keyStatus.down = true
114             this.keyStatus.left = false
115             this.keyStatus.right = false
116         }
117     } else {
118         // 横方向
119         if (horizonDirection < 0) {
120             // left
121             this.keyStatus.up = false
122             this.keyStatus.down = false
123             this.keyStatus.left = true
124             this.keyStatus.right = false
125         } else if (0 <= horizonDirection) {
126             // right
127             this.keyStatus.up = false
128             this.keyStatus.down = false
129             this.keyStatus.left = false
130             this.keyStatus.right = true
131         }
132     }
133 }
134 }
135 //ぶよ設置確認
136 static createNewPuyo () {
137     // ぶよぶよが置けるかどうか、1番上の段の左から3つ目を確認する
138     if(Stage.board[0][2]) {
139         // 空白でない場合は新しいぶよを置けない
140         return false;
141     }
142     // 新しいぶよの色を決める
143     const puyoColors = Math.max(1, Math.min(5, Config.puyoColors));
144     this.centerPuyo = Math.floor(Math.random() * puyoColors) + 1;
145     this.movablePuyo = Math.floor(Math.random() * puyoColors) + 1;
146     // 新しいぶよ画像を作成する
147     this.centerPuyoElement = PuyoImage.getPuyo(this.centerPuyo);
148     this.movablePuyoElement = PuyoImage.getPuyo(this.movablePuyo);
149     Stage.stageElement.appendChild(this.centerPuyoElement);
150     Stage.stageElement.appendChild(this.movablePuyoElement);

```

```

151 // ぶよの初期配置を定める
152 this.puyoStatus = {
153     x: 2, // 中心ぶよの位置: 左から2列目
154     y: -1, // 画面上部ギリギリから出てくる
155     left: 2 * Config.puyoImgWidth,
156     top: -1 * Config.puyoImgHeight,
157     dx: 0, // 動くぶよの相対位置: 動くぶよは上方向にある
158     dy: -1,
159     rotation: 90 // 動くぶよの角度は90度(上向き)
160 };
161 // 接地時間はゼロ
162 this.groundFrame = 0;
163 // ぶよを描画
164 this.setPuyoPosition();
165 return true;
166 }
167
168 static setPuyoPosition () {
169     this.centerPuyoElement.style.left = this.puyoStatus.left + 'px';
170     this.centerPuyoElement.style.top = this.puyoStatus.top + 'px';
171     const x = this.puyoStatus.left + Math.cos(this.puyoStatus.rotation * Math.PI / 180) * Config.puyoImgWidth;
172     const y = this.puyoStatus.top - Math.sin(this.puyoStatus.rotation * Math.PI / 180) * Config.puyoImgHeight;
173     this.movablePuyoElement.style.left = x + 'px';
174     this.movablePuyoElement.style.top = y + 'px';
175 }
176
177 static falling (isDownPressed) {
178     // 現状の場所の下にブロックがあるかどうか確認する
179     let isBlocked = false;
180     let x = this.puyoStatus.x;
181     let y = this.puyoStatus.y;
182     let dx = this.puyoStatus.dx;
183     let dy = this.puyoStatus.dy;
184     if(y + 1 >= Config.stageRows || Stage.board[y + 1][x] || (y + dy + 1 >= 0 && (y + dy + 1 >= Config.stageRows
185         || Stage.board[y + dy + 1][x + dx]))) {
186         isBlocked = true;
187     }
188     if(!isBlocked) {
189         // 下にブロックがないなら自由落下してよい。プレイヤー操作中の自由落下処理をする
190         this.puyoStatus.top += Config.playerFallingSpeed;
191         if(isDownPressed) {
192             // 下キーが押されているならもっと加速する
193             this.puyoStatus.top += Config.playerDownSpeed;
194         }
195         if(Math.floor(this.puyoStatus.top / Config.puyoImgHeight) != y) {
196             // ブロックの境を超えたので、再チェックする
197             // 下キーが押されていたら、得点を加算する
198             if(isDownPressed) {
199                 Score.addScore(1);
200             }
201             y += 1;

```

```

201     this.puyoStatus.y = y;
202     if(y + 1 >= Config.stageRows || Stage.board[y + 1][x] || (y + dy + 1 >= 0 && (y + dy + 1 >= Config.
stageRows || Stage.board[y + dy + 1][x + dx]))) {
203         isBlocked = true;
204     }
205     if(!isBlocked) {
206         // 境を超えたが特に問題はなかった。次回も自由落下を続ける
207         this.groundFrame = 0;
208         return;
209     } else {
210         // 境を超えたらブロックにぶつかった。位置を調節して、接地を開始する
211         this.puyoStatus.top = y * Config.puyoImgHeight;
212         this.groundFrame = 1;
213         return;
214     }
215     } else {
216         // 自由落下で特に問題がなかった。次回も自由落下を続ける
217         this.groundFrame = 0;
218         return;
219     }
220 }
221 if(this.groundFrame == 0) {
222     // 初接地である。接地を開始する
223     this.groundFrame = 1;
224     return;
225 } else {
226     this.groundFrame++;
227     if(this.groundFrame > Config.playerGroundFrame) {
228         return true;
229     }
230 }
231
232 }
233 static playing(frame) {
234     // まず自由落下を確認する
235     // 下キーが押されていた場合、それ込みで自由落下させる
236     if(this.falling(this.keyStatus.down)) {
237         // 落下が終わっていたら、ぷよを固定する
238         this.setPuyoPosition();
239         return 'fix';
240     }
241     this.setPuyoPosition();
242     if(this.keyStatus.right || this.keyStatus.left) {
243         // 左右の確認をする
244         const cx = (this.keyStatus.right) ? 1 : -1;
245         const x = this.puyoStatus.x;
246         const y = this.puyoStatus.y;
247         const mx = x + this.puyoStatus.dx;
248         const my = y + this.puyoStatus.dy;
249         // その方向にブロックがないことを確認する
250         // まずは自分の左右を確認

```

```

251     let canMove = true;
252     if(y < 0 || x + cx < 0 || x + cx >= Config.stageCols || Stage.board[y][x + cx]) {
253         if(y >= 0) {
254             canMove = false;
255         }
256     }
257     if(my < 0 || mx + cx < 0 || mx + cx >= Config.stageCols || Stage.board[my][mx + cx]) {
258         if(my >= 0) {
259             canMove = false;
260         }
261     }
262     // 接地していない場合は、さらに1個下のブロックの左右も確認する
263     if(this.groundFrame === 0) {
264         if(y + 1 < 0 || x + cx < 0 || x + cx >= Config.stageCols || Stage.board[y + 1][x + cx]) {
265             if(y + 1 >= 0) {
266                 canMove = false;
267             }
268         }
269         if(my + 1 < 0 || mx + cx < 0 || mx + cx >= Config.stageCols || Stage.board[my + 1][mx + cx]) {
270             if(my + 1 >= 0) {
271                 canMove = false;
272             }
273         }
274     }
275
276     if(canMove) {
277         // 動かすことが出来るので、移動先情報をセットして移動状態にする
278         this.actionStartFrame = frame;
279         this.moveSource = x * Config.puyoImgWidth;
280         this.moveDestination = (x + cx) * Config.puyoImgWidth;
281         this.puyoStatus.x += cx;
282         return 'moving';
283     }
284     } else if(this.keyStatus.up) {
285         // 回転を確認する
286         // 回せるかどうかは後で確認。まわすぞ
287         const x = this.puyoStatus.x;
288         const y = this.puyoStatus.y;
289         const mx = x + this.puyoStatus.dx;
290         const my = y + this.puyoStatus.dy;
291         const rotation = this.puyoStatus.rotation;
292         let canRotate = true;
293
294         let cx = 0;
295         let cy = 0;
296         if(rotation === 0) {
297             // 右から上には100% 確実に回せる。何もしない
298         } else if(rotation === 90) {
299             // 上から左に回すときに、左にブロックがあれば右に移動する必要があるのでまず確認する
300             if(y + 1 < 0 || x - 1 < 0 || x - 1 >= Config.stageCols || Stage.board[y + 1][x - 1]) {

```

```

301         if(y + 1 >= 0) {
302             // ブロックがある。右に1個ずれる
303             cx = 1;
304         }
305     }
306     // 右にずれる必要がある時、右にもブロックがあれば回転出来ないの確認する
307     if(cx === 1) {
308         if(y + 1 < 0 || x + 1 < 0 || y + 1 >= Config.stageRows || x + 1 >= Config.stageCols || Stage.
            board[y + 1][x + 1]) {
309             if(y + 1 >= 0) {
310                 // ブロックがある。回転出来なかった
311                 canRotate = false;
312             }
313         }
314     }
315     } else if(rotation === 180) {
316         // 左から下に回す時には、自分の下か左下にブロックがあれば1個上に引き上げる。まず下を確認する
317         if(y + 2 < 0 || y + 2 >= Config.stageRows || Stage.board[y + 2][x]) {
318             if(y + 2 >= 0) {
319                 // ブロックがある。上に引き上げる
320                 cy = -1;
321             }
322         }
323         // 左下も確認する
324         if(y + 2 < 0 || y + 2 >= Config.stageRows || x - 1 < 0 || Stage.board[y + 2][x - 1]) {
325             if(y + 2 >= 0) {
326                 // ブロックがある。上に引き上げる
327                 cy = -1;
328             }
329         }
330     } else if(rotation === 270) {
331         // 下から右に回すときは、右にブロックがあれば左に移動する必要があるのでまず確認する
332         if(y + 1 < 0 || x + 1 < 0 || x + 1 >= Config.stageCols || Stage.board[y + 1][x + 1]) {
333             if(y + 1 >= 0) {
334                 // ブロックがある。左に1個ずれる
335                 cx = -1;
336             }
337         }
338         // 左にずれる必要がある時、左にもブロックがあれば回転出来ないの確認する
339         if(cx === -1) {
340             if(y + 1 < 0 || x - 1 < 0 || x - 1 >= Config.stageCols || Stage.board[y + 1][x - 1]) {
341                 if(y + 1 >= 0) {
342                     // ブロックがある。回転出来なかった
343                     canRotate = false;
344                 }
345             }
346         }
347     }
348 }
349 if(canRotate) {
350     // 上に移動する必要があるときは、一気にあげてしまう

```

```

351         if(cy === -1) {
352             if(this.groundFrame > 0) {
353                 // 接地しているなら1段引き上げる
354                 this.puyoStatus.y -= 1;
355                 this.groundFrame = 0;
356             }
357             this.puyoStatus.top = this.puyoStatus.y * Config.puyoImgHeight;
358         }
359         // 回すことが出来るので、回転後の情報をセットして回転状態にする
360         this.actionStartFrame = frame;
361         this.rotateBeforeLeft = x * Config.puyoImgHeight;
362         this.rotateAfterLeft = (x + cx) * Config.puyoImgHeight;
363         this.rotateFromRotation = this.puyoStatus.rotation;
364         // 次の状態を先に設定しておく
365         this.puyoStatus.x += cx;
366         const distRotation = (this.puyoStatus.rotation + 90) % 360;
367         const dCombi = [[1, 0], [0, -1], [-1, 0], [0, 1]][distRotation / 90];
368         this.puyoStatus.dx = dCombi[0];
369         this.puyoStatus.dy = dCombi[1];
370         return 'rotating';
371     }
372 }
373 return 'playing';
374 }
375 static moving(frame) {
376     // 移動中も自然落下はさせる
377     this.falling();
378     const ratio = Math.min(1, (frame - this.actionStartFrame) / Config.playerMoveFrame);
379     this.puyoStatus.left = ratio * (this.moveDestination - this.moveSource) + this.moveSource;
380     this.setPuyoPosition();
381     if(ratio === 1) {
382         return false;
383     }
384     return true;
385 }
386 static rotating(frame) {
387     // 回転中も自然落下はさせる
388     this.falling();
389     const ratio = Math.min(1, (frame - this.actionStartFrame) / Config.playerRotateFrame);
390     this.puyoStatus.left = (this.rotateAfterLeft - this.rotateBeforeLeft) * ratio + this.rotateBeforeLeft;
391     this.puyoStatus.rotation = this.rotateFromRotation + ratio * 90;
392     this.setPuyoPosition();
393     if(ratio === 1) {
394         this.puyoStatus.rotation = (this.rotateFromRotation + 90) % 360;
395         return false;
396     }
397     return true;
398 }
399
400 static fix() {

```

```
401 // 現在のぷよをステージ上に配置する
402 const x = this.puyoStatus.x;
403 const y = this.puyoStatus.y;
404 const dx = this.puyoStatus.dx;
405 const dy = this.puyoStatus.dy;
406 if(y >= 0) {
407     // 画面外のぷよは消してしまう
408     Stage.setPuyo(x, y, this.centerPuyo);
409     Stage.puyoCount++;
410 }
411 if(y + dy >= 0) {
412     // 画面外のぷよは消してしまう
413     Stage.setPuyo(x + dx, y + dy, this.movablePuyo);
414     Stage.puyoCount++;
415 }
416 // 操作作用に作成したぷよ画像を消す
417 Stage.stageElement.removeChild(this.centerPuyoElement);
418 Stage.stageElement.removeChild(this.movablePuyoElement);
419 this.centerPuyoElement = null;
420 this.movablePuyoElement = null;
421 }
422
423 static batankyu() {
424     if (this.keyStatus.up) {
425         location.reload();
426     }
427 }
428 }
```


stage.js

```
1 class Stage {
2     // static stageElement;
3     // static scoreElement;
4     // static zenkeshiImage;
5     // static board;
6     // static puyoCount;
7     // static fallingPuyoList = [];
8     // static eraseStartFrame;
9     // static erasingPuyoInfolist = [];
10
11     static initialize() {
12         // HTML からステージの元となる要素を取得し、大きさを設定する
13         const stageElement = document.getElementById("stage");
14         stageElement.style.width = Config.puyoImgWidth * Config.stageCols + 'px';
15         stageElement.style.height = Config.puyoImgHeight * Config.stageRows + 'px';
16         stageElement.style.backgroundColor = Config.stageBackgroundColor;
17         this.stageElement = stageElement;
18
19         const zenkeshiImage = document.getElementById("zenkeshi");
20         zenkeshiImage.width = Config.puyoImgWidth * 6;
21         zenkeshiImage.style.position = 'absolute';
22         zenkeshiImage.style.display = 'none';
23         this.zenkeshiImage = zenkeshiImage;
24         stageElement.appendChild(zenkeshiImage);
25
26         const scoreElement = document.getElementById("score");
27         scoreElement.style.backgroundColor = Config.scoreBackgroundColor;
28         scoreElement.style.top = Config.puyoImgHeight * Config.stageRows + 'px';
29         scoreElement.style.width = Config.puyoImgWidth * Config.stageCols + 'px';
30         scoreElement.style.height = Config.fontHeight + "px";
31         this.scoreElement = scoreElement;
32
33         // メモリを準備する
34         this.board = [
35             [0, 0, 0, 0, 0, 0],
36             [0, 0, 0, 0, 0, 0],
37             [0, 0, 0, 0, 0, 0],
38             [0, 0, 0, 0, 0, 0],
39             [0, 0, 0, 0, 0, 0],
40             [0, 0, 0, 0, 0, 0],
41             [0, 0, 0, 0, 0, 0],
42             [0, 0, 0, 0, 0, 0],
43             [0, 0, 0, 0, 0, 0],
44             [0, 0, 0, 0, 0, 0],
45             [0, 0, 0, 0, 0, 0],
46             [0, 0, 0, 0, 0, 0],
47         ];
48         let puyoCount = 0;
49         for(let y = 0; y < Config.stageRows; y++) {
50             const line = this.board[y] || (this.board[y] = []);
```

```

51     for(let x = 0; x < Config.stageCols; x++) {
52         const puyo = line[x];
53         if(puyo >= 1 && puyo <= 5) {
54             // line[x] = {puyo: puyo, element: this.setPuyo(x, y, puyo)};
55             this.setPuyo(x, y, puyo);
56             puyoCount++;
57         } else {
58             line[x] = null;
59         }
60     }
61 }
62 this.puyoCount = puyoCount;
63 }
64
65 // 画面とメモリ両方に puyo をセットする
66 static setPuyo(x, y, puyo) {
67     // 画像を作成し配置する
68     const puyoImage = PuyoImage.getPuyo(puyo);
69     puyoImage.style.left = x * Config.puyoImgWidth + "px";
70     puyoImage.style.top = y * Config.puyoImgHeight + "px";
71     this.stageElement.appendChild(puyoImage);
72     // メモリにセットする
73     this.board[y][x] = {
74         puyo: puyo,
75         element: puyoImage
76     }
77 }
78
79 // 自由落下をチェックする
80 static checkFall() {
81     this.fallingPuyoList.length = 0;
82     let isFalling = false;
83     // 下の行から上の行を見ていく
84     for(let y = Config.stageRows - 2; y >= 0; y--) {
85         const line = this.board[y];
86         for(let x = 0; x < line.length; x++) {
87             if(!this.board[y][x]) {
88                 // このマスにぷよがなければ次
89                 continue;
90             }
91             if(!this.board[y + 1][x]) {
92                 // このぷよは落ちるので、取り除く
93                 let cell = this.board[y][x];
94                 this.board[y][x] = null;
95                 let dst = y;
96                 while(dst + 1 < Config.stageRows && this.board[dst + 1][x] == null) {
97                     dst++;
98                 }
99                 // 最終目的地に置く
100                this.board[dst][x] = cell;

```

```

101 // 落ちるリストに入れる
102 this.fallingPuyoList.push({
103     element: cell.element,
104     position: y * Config.puyoImgHeight,
105     destination: dst * Config.puyoImgHeight,
106     falling: true
107 });
108 // 落ちるものがあったことを記録しておく
109 isFalling = true;
110 }
111 }
112 }
113 return isFalling;
114 }
115 // 自由落下させる
116 static fall() {
117     let isFalling = false;
118     for(const fallingPuyo of this.fallingPuyoList) {
119         if(!fallingPuyo.falling) {
120             // すでに自由落下が終わっている
121             continue;
122         }
123         let position = fallingPuyo.position;
124         position += Config.freeFallingSpeed;
125         if(position >= fallingPuyo.destination) {
126             // 自由落下終了
127             position = fallingPuyo.destination;
128             fallingPuyo.falling = false;
129         } else {
130             // まだ落下しているぷよがあることを記録する
131             isFalling = true;
132         }
133         // 新しい位置を保存する
134         fallingPuyo.position = position;
135         // ぷよを動かす
136         fallingPuyo.element.style.top = position + 'px';
137     }
138     return isFalling;
139 }
140
141 // 消せるかどうか判定する
142 static checkErase(startFrame) {
143     this.eraseStartFrame = startFrame;
144     this.erasingPuyoInfoList.length = 0;
145
146     // 何色のぷよを消したかを記録する
147     const erasedPuyoColor = {};
148
149     // 隣接ぷよを確認する関数内関数を作成
150     const sequencePuyoInfoList = [];

```

```

151     const existingPuyoInfoList = [];
152     const checkSequentialPuyo = (x, y) => {
153         // ぶよがあるか確認する
154         const orig = this.board[y][x];
155         if(!orig) {
156             // ないなら何もしない
157             return;
158         }
159         // あるなら一旦退避して、メモリ上から消す
160         const puyo = this.board[y][x].puyo;
161         sequencePuyoInfoList.push({
162             x: x,
163             y: y,
164             cell: this.board[y][x]
165         });
166         this.board[y][x] = null;
167
168         // 四方向の周囲ぶよを確認する
169         const direction = [[0, 1], [1, 0], [0, -1], [-1, 0]];
170         for(let i = 0; i < direction.length; i++) {
171             const dx = x + direction[i][0];
172             const dy = y + direction[i][1];
173             if(dx < 0 || dy < 0 || dx >= Config.stageCols || dy >= Config.stageRows) {
174                 // ステージの外にはみ出た
175                 continue;
176             }
177             const cell = this.board[dy][dx];
178             if(!cell || cell.puyo !== puyo) {
179                 // ぶよの色が違う
180                 continue;
181             }
182             // そのぶよのまわりのぶよも消せるか確認する
183             checkSequentialPuyo(dx, dy);
184         };
185     };
186
187     // 実際に削除できるかの確認を行う
188     for(let y = 0; y < Config.stageRows; y++) {
189         for(let x = 0; x < Config.stageCols; x++) {
190             sequencePuyoInfoList.length = 0;
191             const puyoColor = this.board[y][x] && this.board[y][x].puyo;
192             checkSequentialPuyo(x, y);
193             if(sequencePuyoInfoList.length == 0 || sequencePuyoInfoList.length < Config.erasePuyoCount) {
194                 // 連続して並んでいる数が足りなかったので消さない
195                 if(sequencePuyoInfoList.length) {
196                     // 退避していたぶよを消さないリストに追加する
197                     existingPuyoInfoList.push(...sequencePuyoInfoList);
198                 }
199             } else {
200                 // これらは消して良いので消すリストに追加する

```

```
201         this.erasingPuyoInfoList.push(...sequencePuyoInfoList);
202         erasedPuyoColor[puyoColor] = true;
203     }
204 }
205 }
206 this.puyoCount -= this.erasingPuyoInfoList.length;
207
208 // 消さないリストに入っていたぷよをメモリに復帰させる
209 for(const info of existingPuyoInfoList) {
210     this.board[info.y][info.x] = info.cell;
211 }
212
213 if(this.erasingPuyoInfoList.length) {
214     // もし消せるならば、消えるぷよの個数と色の情報をまとめて返す
215     return {
216         piece: this.erasingPuyoInfoList.length,
217         color: Object.keys(erasedPuyoColor).length
218     };
219 }
220 return null;
221 }
222 // 消すアニメーションをする
223 static erasing(frame) {
224     const elapsedFrame = frame - this.eraseStartFrame;
225     const ratio = elapsedFrame / Config.eraseAnimationDuration;
226     if(ratio > 1) {
227         // アニメーションを終了する
228         for(const info of this.erasingPuyoInfoList) {
229             var element = info.cell.element;
230             this.stageElement.removeChild(element);
231         }
232         return false;
233     } else if(ratio > 0.75) {
234         for(const info of this.erasingPuyoInfoList) {
235             var element = info.cell.element;
236             element.style.display = 'block';
237         }
238         return true;
239     } else if(ratio > 0.50) {
240         for(const info of this.erasingPuyoInfoList) {
241             var element = info.cell.element;
242             element.style.display = 'none';
243         }
244         return true;
245     } else if(ratio > 0.25) {
246         for(const info of this.erasingPuyoInfoList) {
247             var element = info.cell.element;
248             element.style.display = 'block';
249         }
250         return true;
```

```

251     } else {
252         for(const info of this.erasingPuyoInfoList) {
253             var element = info.cell.element;
254             element.style.display = 'none';
255         }
256         return true;
257     }
258 }
259
260 static showZenkeshi() {
261     // 全消しを表示する
262     this.zenkeshiImage.style.display = 'block';
263     this.zenkeshiImage.style.opacity = '1';
264     const startTime = Date.now();
265     const startTop = Config.puyoImgHeight * Config.stageRows;
266     const endTop = Config.puyoImgHeight * Config.stageRows / 3;
267     const animation = () => {
268         const ratio = Math.min((Date.now() - startTime) / Config.zenkeshiDuration, 1);
269         this.zenkeshiImage.style.top = (endTop - startTop) * ratio + startTop + 'px';
270         if(ratio !== 1) {
271             requestAnimationFrame(animation);
272         }
273     };
274     animation();
275 }
276 static hideZenkeshi() {
277     // 全消しを消去する
278     const startTime = Date.now();
279     const animation = () => {
280         const ratio = Math.min((Date.now() - startTime) / Config.zenkeshiDuration, 1);
281         this.zenkeshiImage.style.opacity = String(1 - ratio);
282         if(ratio !== 1) {
283             requestAnimationFrame(animation);
284         } else {
285             this.zenkeshiImage.style.display = 'none';
286         }
287     };
288     animation();
289 }
290 }
291 Stage.fallingPuyoList = [];
292 Stage.erasingPuyoInfoList = [];
293

```

score.js

```
1 class Score {
2   // static fontTemplateList = [];
3   // static fontLength;
4   // static score = 0;
5
6   static initialize() {
7     this.fontTemplateList = [];
8     let fontWidth = 0;
9     for(let i = 0; i < 10; i++) {
10      const fontImage = document.getElementById(`font${i}`);
11      if(fontWidth === 0) {
12        fontWidth = fontImage.width / fontImage.height * Config.fontHeight;
13      }
14      fontImage.height = Config.fontHeight;
15      fontImage.width = fontWidth;
16      this.fontTemplateList.push(fontImage);
17    }
18
19    this.fontLength = Math.floor(Config.stageCols * Config.puyoImgWidth / this.fontTemplateList[0].width);
20    this.score = 0;
21    this.showScore();
22  }
23  static showScore () {
24    let score = this.score;
25    const scoreElement = Stage.scoreElement;
26    // まず最初に、scoreElement の中身を空っぽにする
27    while(scoreElement.firstChild) {
28      scoreElement.removeChild(scoreElement.firstChild);
29    }
30    // スコアを下の桁から埋めていく
31    for(let i = 0; i < this.fontLength; i++) {
32      // 10で割ったあまりを求めて、一番下の桁を取り出す
33      const number = score % 10;
34      // 一番うしろに追加するのではなく、一番前に追加することで、スコアの並びを数字と同じようにする
35      scoreElement.insertBefore(this.fontTemplateList[number].cloneNode(true), scoreElement.firstChild);
36      // 10 で割って次の桁の準備をしておく
37      score = Math.floor(score / 10);
38    }
39  }
40  static calculateScore (rensa, piece, color) {
41    rensa = Math.min(rensa, Score.rensaBonus.length - 1);
42    piece = Math.min(piece, Score.pieceBonus.length - 1);
43    color = Math.min(color, Score.colorBonus.length - 1);
44    let scale = Score.rensaBonus[rensa] + Score.pieceBonus[piece] + Score.colorBonus[color];
45    if(scale === 0) {
46      scale = 1;
47    }
48    this.addScore(scale * piece * 10);
49  }
50  static addScore (score) {
```

51	this.score += score;
52	this.showScore();
53	}
54	};
55	
56	Score.rensaBonus = [0, 8, 16, 32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672];
57	Score.pieceBonus = [0, 0, 0, 0, 2, 3, 4, 5, 6, 7, 10, 10];
58	Score.colorBonus = [0, 0, 3, 6, 12, 24];
59	

config.js

1	// 設定を記載しておくクラス
2	class Config {
3	}
4	Config.puyoImgWidth = 40; // ぶよぶよ画像の幅
5	Config.puyoImgHeight = 40; // ぶよぶよ画像の高さ
6	
7	Config.fontHeight = 33;
8	
9	Config.stageCols = 6; // ステージの横の個数
10	Config.stageRows = 12; // ステージの縦の個数
11	
12	// フィールドサイズ追加
13	// 高さが全部入るように調整
14	Config.puyoImgHeight = (window.innerHeight-Config.fontHeight)/Config.stageRows
15	Config.puyoImgWidth = Config.puyoImgHeight;
16	
17	Config.stageBackgroundColor = '#ffffff'; // ステージの背景色
18	Config.scoreBackgroundColor = '#24c0bb'; // スコアの背景色
19	
20	Config.freeFallingSpeed = 16; // 自由落下のスピード
21	Config.erasePuyoCount = 4; // 何個以上揃ったら消えるか
22	Config.eraseAnimationDuration = 30; // 何フレームでぶよを消すか
23	
24	Config.puyoColors = 4; // 何色のぶよを使うか
25	Config.playerFallingSpeed = 0.9; // プレイ中の自然落下のスピード
26	Config.playerDownSpeed = 10; // プレイ中の下キー押下時の落下スピード
27	Config.playerGroundFrame = 20; // 何フレーム接地したらぶよを固定するか
28	Config.playerMoveFrame = 10; // 左右移動に消費するフレーム数
29	Config.playerRotateFrame = 10; // 回転に消費するフレーム数
30	
31	Config.zenkeshiDuration = 150; // 全消し時のアニメーションミリ秒
32	Config.gameOverFrame = 3000; // ゲームオーバー演出のサイクルフレーム
33	