```
1   // Original source code: https://phaser.io/examples/v2/games/sliding-puzzle
2   // Article: http://zofiakorcz.pl/sliding-puzzle-game-phaser
3   // GitHub: https://github.com/Calanthe/PhaserSlidingPuzzle
4   // Live Demo v2.3.0: http://zofiakorcz.pl/PhaserSlidingPuzzle/
5
6   // ========================================================
7   // Acknowledgments:
8   // ========================================================
9   console.log ("%c %c %c|     Phaser 3 Game Prototyping Starter Kit
    |%c %c ","background: #00bff3","background: #0072bc","color: #ffffff;
    background: #003471","background: #0072bc","background: #00bff3");
10  console.log ("%c %c %c|         Slider Puzzle Game Mechanics!
    |%c %c ","background: #00bff3","background: #0072bc","color: #ffffff;
    background: #003471","background: #0072bc","background: #00bff3");
11  console.log ("%c %c %c|      Copyright \u00A9 1974-2017, Stephen
    Gose.          |%c %c ","background: #00bff3","background:
    #0072bc","color: #ffffff; background: #003471","background:
    #0072bc","background: #00bff3");
12  console.log ("%c %c %c|                All rights reserved.
    |%c %c ","background: #00bff3","background: #0072bc","color: #ffffff;
    background: #003471","background: #0072bc","background: #00bff3");
13  console.log ("%c %c %c|  Book available at: https://leanpub.com/p3gskc
    |%c %c ","background: #00bff3","background: #0072bc","color: #ffffff;
    background: #003471","background: #0072bc","background: #00bff3");
14  console.log ("%c %c %c|  \u2665\u2665\u2665\u2665\u2665 -> $120
    License(s) included in book!       |%c %c ","background:
    #00bff3","background: #0072bc","color: #ffffff; background:
    #003471","background: #0072bc","background: #00bff3");
15
16  // ========================================================
17  function gameMechanics(GAMEAPP,title){
18
19      console.log("%c %c %c|              "+title+" launched
        |%c %c ","background: #00bff3","background: #0072bc","color: #ffffff;
        background: #003471","background: #0072bc","background: #00bff3");
20      console.log("Game Mechanism === Extensive? "+Object.isExtensible(
        GAMEAPP ));
21      console.log(Object.values(GAMEAPP));
22      console.log(Object.getPrototypeOf(GAMEAPP));
23      console.log(Object.getOwnPropertyDescriptors(GAMEAPP));
24
25  };
26
27  // ========================================================
28  // pure JavaScript Game Scaling:
29  //  - also available within Phaser III config settings
30  // ========================================================
31  function resize() {
32      var canvas = document.querySelector("canvas");
33      var windowWidth = window.innerWidth;
34      var windowHeight = window.innerHeight;
35      // window ratio discussed in Phaser III Game Design
36      var windowRatio = windowWidth / windowHeight;
37      var gameRatio = config.width / config.height;
38      //console.info("Win Ration: "+windowRatio+" | Game Ration:
        "+gameRatio);
39      if(windowRatio < gameRatio){
40          canvas.style.width = windowWidth + "px";
41          canvas.style.height = (windowWidth / gameRatio) + "px";
```



```
42          } else {
43              canvas.style.width = (windowHeight * gameRatio) + "px";
44              canvas.style.height = windowHeight + "px";
45          } // End window adjustments and sizing
46      };
47
48      function swopPieces(gameObject){
49          console.log("Pz Piece clicked.")
50
51      };
52      // ================================================
53      // Game global config object
54      // ================================================
55      // creates our Phaser III Game configurations.
56      // JS object literals are NOT hoisted.
57      var config = {
58          //dozens of configurations parameters available.
59          type: Phaser.AUTO,  // .AUTO, .WEBGL or .Canvas
60          width:    1200,     // x width - best results use Golden Ratio
61          height:   1200,     // y width - best results use Golden Ratio
62
63          title: 'Phaser3 Game Prototyping Starter Kit \n',   //Game Title
64          url: '\nhttp://makingbrowsergames.com/p3gp-book/',  //Game URL location
65          //http://semver.org/spec/v1.0.0.html
66          version: 'ersion  1.0.0.0 (semver) 20171103       \n',
67          input: {
68              gamepad: false, // not interested in game pad input
69              keyboard: true,
70              mouse: true,
71              touch: true,
72              // Breaking Change: v3.16+
73              // queue: true  // use pre-v3.16 legacy Input Manager event queue
74          },
75          backgroundColor: 0x222222, //Custom RGB color. (0x000 - 0xFFF)
76          physics: {
77              default: 'arcade'
78          },
79          dom: {
80              createContainer: true
81          },
82          scene:  SliderPuzzle,
83          //parent: document.body,
84          //parent:   "game-area",
85          pixelArt: false,
86          antialias: true
87      };
88      let setOptions = {
89          tileSize:       100,    // set for artwork imported
90          swapSpeed:      200,    // tile's tween speed; not need in Draw-3
            versions
91          // rotateSpeed: 100,    // for gameboard spin version; see book
92          fallSpeed:      100,    // tile's tween speed
93          destroySpeed:   200,    // removal speed
94          boardOffset: {          // set gameBoard margins
95              x: 20,
96              y: 20
97          }
98      }; // End game settings; TODO expose to user?
99
```

```
100  var game = {};  // Phaser Game object
101
102  function SliderPuzzle(game) {
103      ALLOW_CLICK = 0;
104      TWEENING = 1;
105      // These are all set in the startPuzzle function
106      this.rows = 0;
107      this.columns = 0;
108
109      // The width and height of each piece in the puzzle.
110      // Again, this is set automatically in startPuzzle.
111      this.pieceWidth = 0;
112      this.pieceHeight = 0;
113
114      // as groups? Difficult! containers? NO, grids? NO, tilemaps? MMM!
115      this.pzGrp = null;  // visual puzzle Group (pzGrp) - Model View
116      this.puzzle = [];   // game mechanics - data structure Model
117      this.spacer = null; // last piece become the "black spacer"
118
119      // The speed at which the pzGrp slide, and the tweens they use
120      this.slideSpeed = setOptions.swapSpeed; // originally 250;
121      this.slideEase = 'Sine';
122
123      // The number of iterations the puzzle walker will cycle when
124      // scrambling the puzzle pieces. 10 is a simple and easy puzzle, but
125      // set it higher for more difficult puzzle sliders.
126      this.iterations = 10;
127
128      // The speed at which pzGrp are shuffled from initialization. This
         allows
129      // the player to see the puzzle before trying to solve it. However if
130      // you don't want this, just set the speed to zero and it'll appear
131      // instantly 'scrambled' as the previous versions did.
132      this.shuffleSpeed = setOptions.fallSpeed;   // originally 80;
133      this.shuffleEase = 'Linear';
134
135      this.lastMove = null;
136
137      // The image in the Cache to be used for the puzzle.
138      // Set in the startPuzzle function.
139      this.photo = '';
140
141      this.action = this.ALLOW_CLICK;
142
143  }
144
145  SliderPuzzle.prototype = {
146      // Phaser Essential Function (life-cycle)
147      preload: function(){
148          console.log("loading play state");
149          // this.load.path = 'assets/';
150          // default image is .png
151          this.load.image('photo1', 'assets/jt-pic3-3.jpg');
152          //this.load.images([ 'photo1', 'photo2', 'photo3' ]);
153      },
154      //
155      // ========================================================
156      // A simple background color, and then the puzzle is created in
         startPuzzle.
```



```
157         create: function(){
158             console.log("starting play state");
159 
160             //this.pic =
                this.add.image(config.width/2,config.height/2-50,'photo1').setOrigi
                n(0.5,0.5);
161             //console.log(this.pic);
162             // slider for puzzle images
163             this.startPuzzle('photo1', 3, 3);
164 
165             pointer = this.input.activePointer;
166             // this.input.mouse.disableContextMenu();   // security measure
167 
168         },
169         //
170         // ========================================================
171         update: function(){
172 
173         },  //place " , " If using more supporting function below
174 
175         // ========================================================
176         // ----------------------------------------
177         // Supporting game Function & Classes
178         // ----------------------------------------
179         // ========================================================
180         /**
181          * This function is responsible for building the puzzle.
182          * It takes an Image key and a width and height of the puzzle
183          * (in pzGrp, not pixels).
184          * Read the comments within this function to find out what happens.
185          */
186         startPuzzle: function (key, rows, columns) {
187             // Get the size of this image
188             // P2: var photoWidth = this.cache.getImage(key).width;
189             // P2: var photoWidth = this.cache.getImage(key).width;
190             // P3
191             this.photo = this.textures.get(key);
192             this.photo.getSourceImage();
193             var photoWidth = this.photo.source[0].width;
194             var photoHeight = this.photo.source[0].height;
195             console.log("Image is : w: "+photoWidth+" x h: "+photoHeight);
196 
197             //  The shards of the puzzle, in pzGrp (not pixels)
198             this.rows = rows;
199             this.columns = columns;
200             console.log(" - divided into "+rows+" rows by "+columns+"
                columns");
201 
202             //  Create our sliding pzGrp
203             //  Each piece will be this size:
204             this.pieceWidth = Math.floor(photoWidth / this.rows);
205             this.pieceHeight = Math.floor(photoHeight / this.columns);
206             console.log(" - each shard is "+this.pieceWidth+" width by
                "+this.pieceHeight+" high");
207 
208             //  A Puzzle Group to hold the shards
209             if (this.pzGrp){
210                 //  Group already exists? Destroy the contents
211                 this.pzGrp.destroy(true, true);
```

```
212             } else {
213                 this.pzGrp = this.add.group();
214             }
215 
216             // Center the Puzzle
217             //this.pzGrp.x = Math.floor((config.width - photoWidth) /
                2);       // px
218             this.pzGrp.x = 50;
219             //this.pzGrp.y = Math.floor((config.height - photoHeight) / 2);
                // px
220             this.pzGrp.y = 50;
221 
222             //  Loop through the image and create a new shards piece for the
                puzzle.
223             var ndx = 0;
224             for (var j = 0; j < this.rows; j++) {   // (Y) vertical / down
225                 for (var i = 0; i < this.columns; i++) {    // (X) horizontal
                    / across
226 
227                     // P2: var piece = this.pzGrp.create(i * this.pieceWidth,
                        j * this.pieceHeight, key);
228                     // P3: var piece =
                        this.add.image(config.width/2,config.height/2,'photo1').set
                        Origin(0.5,0.5);
229                     //
                        https://stackoverflow.com/questions/54858223/how-to-store-s
                        prites-in-an-array-in-phaser-3
230                     // group.getChildren() returns an array
231                     // -----------------------------------------
232                     //
                        https://rexrainbow.github.io/phaser3-rex-notes/docs/site/te
                        xture/#texture
233                     //
                        https://rexrainbow.github.io/phaser3-rex-notes/docs/site/te
                        xture/#frame-object
234                     //
                        https://rexrainbow.github.io/phaser3-rex-notes/docs/site/ca
                        nvas-texture/#add-frame
235                     /**Add frame¶
236                         var frame = texture.add(frameName, sourceIndex, x, y,
                            width, height);
237                         texture.add(name, sourceIndex, x, y, width, height);
238                         frameName : The name of this Frame. The name is
                            unique within the Texture.
239                         sourceIndex : The index of the TextureSource that
                            this Frame is a part of.
240                         x : The x coordinate of the top-left of this Frame.
241                         y : The y coordinate of the top-left of this Frame.
242                         width : The width of this Frame.
243                         height : The height of this Frame.
244 
245                     Frame object - Get frame - CutX!¶
246                         var frame = scene.textures.getFrame(key, frame);
247                         Properties¶
248                         frame.source.image : Image of texture source.
249                         frame.cutX : X position within the source image to
                            cut from.
250                         frame.cutY : Y position within the source image to
                            cut from.
```



```
251                     frame.cutWidth : The width of the area in the source
                        image to cut.
252                     frame.cutHeight : The height of the area in the
                        source image to cut.
253                 */
254                 // -----------------------------------------
255                 var x = i * this.pieceWidth;
256                 var y = j * this.pieceHeight;
257
258   console.log(ndx+") x:"+x+" y:"+y);
259                 var piece = this.pzGrp.create(x, y, key);
260                 this.pzGrp.setX(this.pieceWidth,i-1.5);
261                 this.pzGrp.setY(this.pieceHeight,j-1.5);
262
263                 //var piece = this.pzGrp.create(0, 0, key);
264                 piece.setOrigin();
265                 piece.setInteractive();
266
267                 // 3-ways to set data values into gameObjects
268                 // -----------------------------------------
269                 // - set multiple values at once using an object
270                 //   see line 26 of
                    http://labs.phaser.io/edit.html?src=src%5Ccomponents%5Cdata
                    %5Cset%20multiple%20values.js
271                 // - use .setData on object
272                 //   see line 27-29 of
                    http://labs.phaser.io/edit.html?src=src%5Ccomponents%5Cdata
                    %5Cdata%20values.js
273                 //   ref:
                    https://photonstorm.github.io/phaser3-docs/Phaser.GameObjec
                    ts.GameObject.html#setData__anchor
274                 // - use .setDataEnable() on object; Adds a Data Manager
                    component to this Game Object.
275                 //   see line 27 of
                    http://labs.phaser.io/edit.html?src=src%5Ccomponents%5Cdata
                    %5Cchange%20data%20event.js
276                 //   ref:
                    https://phaser.io/examples/v3/view/components/data/get-and-
                    set-data-values
277                 // -----------------------------------------
278                 //piece.setDataEnable();
279                 // The current row and column of the piece
280                 // piece.data.values.row = j;           // grid index
                    location
281                 // piece.data.values.column = i;        // grid index
                    location
282                 piece.setData({name: "C"+j+"_R"+i,index: ndx, row: j,
                    column: i});
283   console.log("--- name: "+piece.getData('name'));
284                 // Store the final resting position of the row and column
                    when the puzzle is solved
285                 piece.data.values.correctRow = j;       // final location
286                 piece.data.values.correctColumn = i;    // final location
287
288                 piece.on('pointerover', function (pointer) {
289                     //console.log(piece.getData('index')+")
                    "+this.getData('name')+": ");
290                     //this.setTint(0xff0000);
291                 });
```

```
292                     piece.on('pointerout', function (pointer) {
293                         //this.clearTint();
294                     });
295 
296                     // Here is how we handle creating each shard. Rather than
                        do something like create
297                     // a BitmapData, and all kinds of extra textures, we just
                        set each shard to have
298                     // the same texture as the original source image. Then we
                        crop it to be the correct
299                     // section (row x column) of the image. This ensures that
                        we've still only have one texture
300                     // bound to the GPU.
301                     // Reference:
                        https://rexrainbow.github.io/phaser3-rex-notes/docs/site/gr
                        oup/
302                     // P2: this.piece.crop(new Phaser.Rectangle(i *
                        this.pieceWidth, j * this.pieceHeight, this.pieceWidth,
                        this.pieceHeight));
303                     // this.input.setHitAreaRectangle(this.piece,i *
                        this.pieceWidth,j * this.pieceHeight,this.pieceWidth,
                        this.pieceHeight);
304                     // P3:
305                     //piece.setCrop(new Phaser.Geom.Rectangle(x, y,
                        this.pieceWidth, this.pieceHeight));
306 
307                     piece.setCrop(x, y, this.pieceWidth,
                        this.pieceHeight).setOrigin(0);
308                     piece.isCropped = true;
309 
310                     // add individual shards into the puzzle Group
311                     // this mixes data structure with View Model (MVVM)
312                     this.pzGrp.add(piece);
313                     // OR push shards into an array; then add array into pzGrp
314                     //this.puzzle.push(piece);      // game mechanic data
                        structure
315 
316                     console.log(piece);
317                     // P2:
318                     //this.piece.events.onInputDown.add(this.checkPiece, this);
319 
320                     if(i==this.columns-1 && j==this.rows-1){
321                         piece.alpha = 0;
322                         // The last piece will be our 'empty space' to start
                            the sliding
323                         // P2: this.spacer =
                            this.pzGrp.getChildAt(this.pzGrp.length - 1);
324                         // P2: this.spacer.alpha = 0;
325                         // P3: getLastNth(nth [, state] [, createIfNull] [,
                            x] [, y] [, key] [, frame] [, visible])
326                         var blackSpacerPiece = this.pzGrp.getLength();
327                         this.spacer = ndx;
328                         //gameMechanics(this.pzGrp,"Puzzle Group")
329                         console.log("Last piece of pzGrp is
                            "+blackSpacerPiece+" turned into an empty space:
                            "+this.spacer+1);
330                     }
331                     ndx++;
332                 }
```

```
333
334             }
335             //this.pzGrp.addMultiple(this.puzzle);  // game mechanisms -
                Model View
336             //console.table(this.puzzle);
337             //console.table(this.pzGrp);
338             //
339             // ================================================
340             // HUD input Button function
341             // ================================================
342             // activate UI
343             //  Mouse Events for puzzle shards
344             this.input.on('pointerover', function (event, gameObjects) {
345                 //console.log(gameObjects.getData('index')+")
                    "+gameObjects.getData('name')+": ");
346                 console.log("- chosen");
347
348             });
349             this.input.on('pointerout', function (event, gameObjects) {
350                 // Design Options: if using tint; clear it
351                 // gameObjects[0].clearTint();
352
353                 console.info("-  off. ");
354
355             });
356             this.input.on('pointerdown',function (event, gameObjects) {
357                 //console.info(Math.floor(pointer.x/100) + " clicked.");
358                 console.info("-  clicked.");
359
360             });
361
362         },
363
364         /**
365          * This shuffles up our puzzle.
366          *
367          * We can't just 'randomize' the tiles, or 50% of the time we'll get an
368          * unsolvable puzzle. So instead lets walk it, making non-repeating
                random moves.
369          */
370         shufflepzGrp: function () {
371
372             //  Push all available moves into this array
373             var moves = [];
374
375             if (this.spacer.data.column > 0 && this.lastMove !== Phaser.DOWN)
376             {
377                 moves.push(Phaser.UP);
378             }
379
380             if (this.spacer.data.column < this.columns - 1 && this.lastMove
                !== Phaser.UP)
381             {
382                 moves.push(Phaser.DOWN);
383             }
384
385             if (this.spacer.data.row > 0 && this.lastMove !== Phaser.RIGHT)
386             {
387                 moves.push(Phaser.LEFT);
```



```
388             }
389
390             if (this.spacer.data.row < this.rows - 1 && this.lastMove !==
                Phaser.LEFT)
391             {
392                 moves.push(Phaser.RIGHT);
393             }
394
395             //  Pick a move at random from the array
396             this.lastMove = this.rnd.pick(moves);
397
398             //  Then move the spacer into the new position
399             switch (this.lastMove)
400             {
401                 case Phaser.UP:
402                     this.swapPiece(this.spacer.data.row,
                        this.spacer.data.column - 1);
403                     break;
404
405                 case Phaser.DOWN:
406                     this.swapPiece(this.spacer.data.row,
                        this.spacer.data.column + 1);
407                     break;
408
409                 case Phaser.LEFT:
410                     this.swapPiece(this.spacer.data.row - 1,
                        this.spacer.data.column);
411                     break;
412
413                 case Phaser.RIGHT:
414                     this.swapPiece(this.spacer.data.row + 1,
                        this.spacer.data.column);
415                     break;
416             }
417
418         },
419
420         /**
421          * Swaps the spacer with the piece in the given row and column.
422          */
423         swapPiece: function (row, column) {
424
425             //  row and column is the new destination of the spacer
426
427             var piece = this.getPiece(row, column);
428
429             var x = this.spacer.x;
430             var y = this.spacer.y;
431
432             piece.data.row = this.spacer.data.row;
433             piece.data.column = this.spacer.data.column;
434
435             this.spacer.data.row = row;
436             this.spacer.data.column = column;
437
438             this.spacer.x = piece.x;
439             this.spacer.y = piece.y;
440
441             //  If we don't want them to watch the puzzle get shuffled, then
```



```
            just
442         //  set the piece to the new position immediately.
443         if (this.shuffleSpeed === 0)
444         {
445             piece.x = x;
446             piece.y = y;
447
448             if (this.iterations > 0)
449             {
450                 //  Any more iterations left? If so, shuffle, otherwise
                    start play
451                 this.iterations--;
452
453                 this.shufflepzGrp();
454             }
455             else
456             {
457                 this.startPlay();
458             }
459         }
460         else
461         {
462             //  Otherwise, tween it into place
463             var tween = this.add.tween(piece).to({ x: x, y: y },
                this.shuffleSpeed, this.shuffleEase, true);
464
465             if (this.iterations > 0)
466             {
467                 //  Any more iterations left? If so, shuffle, otherwise
                    start play
468                 this.iterations--;
469
470                 tween.onComplete.add(this.shufflepzGrp, this);
471             }
472             else
473             {
474                 tween.onComplete.add(this.startPlay, this);
475             }
476         }
477
478     },
479
480     /**
481      * Gets the piece at row and column.
482      */
483     getPiece: function (row, column) {
484
485         for (var i = 0; i < this.pzGrp.children.length; i++)
486         {
487             var piece = this.pzGrp.getChildAt(i);
488
489             if (piece.data.row === row && piece.data.column === column)
490             {
491                 return piece;
492             }
493         }
494
495         return null;
496
```



```
497         },
498
499         /**
500          * Sets the game state to allow the user to click.
501          */
502         startPlay: function () {
503
504             this.action = SliderPuzzle.ALLOW_CLICK;
505
506         },
507
508         /**
509          * Called when the user clicks on any of the puzzle pzGrp.
510          * It first checks to see if the piece is adjacent to the 'spacer',
             and if not, bails out.
511          * If it is, the two pzGrp are swapped by calling `this.slidePiece`.
512          */
513         checkPiece: function (piece) {
514
515             if (this.action !== SliderPuzzle.ALLOW_CLICK)
516             {
517                 return;
518             }
519
520             //  Only allowed if adjacent to the 'spacer'
521             //
522             //  Remember:
523             //
524             //  Columns = vertical (y) axis
525             //  Rows = horizontal (x) axis
526
527             if (piece.data.row === this.spacer.data.row)
528             {
529                 if (this.spacer.data.column === piece.data.column -1)
530                 {
531                     //  Space above the piece?
532                     piece.data.column--;
533
534                     this.spacer.data.column++;
535                     this.spacer.y += this.spacer.height;
536
537                     this.slidePiece(piece, piece.x, piece.y -
                        this.pieceHeight);
538                 }
539                 else if (this.spacer.data.column === piece.data.column + 1)
540                 {
541                     //  Space below the piece?
542                     piece.data.column++;
543
544                     this.spacer.data.column--;
545                     this.spacer.y -= this.spacer.height;
546
547                     this.slidePiece(piece, piece.x, piece.y +
                        this.pieceHeight);
548                 }
549             }
550             else if (piece.data.column === this.spacer.data.column)
551             {
552                 if (this.spacer.data.row === piece.data.row -1)
```

```
553                 {
554                     //  Space to the left of the piece?
555                     piece.data.row--;
556 
557                     this.spacer.data.row++;
558                     this.spacer.x += this.spacer.width;
559 
560                     this.slidePiece(piece, piece.x - this.pieceWidth, piece.y);
561                 }
562                 else if (this.spacer.data.row === piece.data.row + 1)
563                 {
564                     //  Space to the right of the piece?
565                     piece.data.row++;
566 
567                     this.spacer.data.row--;
568                     this.spacer.x -= this.spacer.width;
569 
570                     this.slidePiece(piece, piece.x + this.pieceWidth, piece.y);
571                 }
572             }
573 
574         },
575 
576         /**
577          * Slides the piece into the position previously occupied by the
             spacer.
578          * Uses a tween (see slideSpeed and slideEase for controls).
579          * When complete, calls tweenOver.
580          */
581         slidePiece: function (piece, x, y) {
582 
583             this.action = GAMEAPP.TWEENING;
584 
585             var tween = this.add.tween(piece).to({ x: x, y: y },
                 this.slideSpeed, this.slideEase, true);
586 
587             tween.onComplete.addOnce(this.tweenOver, this);
588 
589         },
590 
591         /**
592          * Called when a piece finishes sliding into place.
593          * First checks if the puzzle is solved. If not, allows the player to
             carry on.
594          */
595         tweenOver: function () {
596 
597             //  Are all the pzGrp in the right place?
598 
599             var outOfSequence = false;
600 
601             this.pzGrp.forEach(function(piece) {
602 
603                 if (piece.data.correctRow !== piece.data.row ||
                     piece.data.correctColumn !== piece.data.column)
604                 {
605                     outOfSequence = true;
606                 }
607 
```

```
608             });
609
610             if (outOfSequence)
611             {
612                 //  Not correct, so let the player carry on.
613                 this.action = GAMEAPP.ALLOW_CLICK;
614             }
615             else
616             {
617                 //  If we get this far then the sequence is correct and the
                    puzzle is solved.
618                 //  Fade the missing piece back in ...
619
620                 var tween = this.add.tween(this.spacer).to({ alpha: 1 },
                    this.slideSpeed * 2, 'Linear', true);
621
622                 //  When the tween finishes we'll let them click to start the
                    next round
623
624                 var _this = this;
625
626                 tween.onComplete.addOnce(function() {
627                     _this.input.onDown.addOnce(_this.nextRound, _this);
628                 });
629
630             }
631
632         },
633
634         /**
635          * Starts the next round of the game.
636          *
637          * In this template it cycles between the 3 pictures, increasing the
                iterations and complexity
638          * as it progresses. But you can replace this with whatever you need
                - perhaps returning to
639          * a main menu to select a new puzzle?
640          */
641         nextRound: function () {
642
643             if (this.photo === 'photo1')
644             {
645                 this.photo = 'photo2';
646                 this.iterations = 20;
647                 this.startPuzzle(this.photo, 4, 4);
648             }
649             else if (this.photo === 'photo2')
650             {
651                 this.photo = 'photo3';
652                 this.iterations = 30;
653                 this.startPuzzle(this.photo, 5, 5);
654             }
655             else
656             {
657                 //  Back to the start again
658                 this.photo = 'photo1';
659                 this.iterations = 10;
660                 this.startPuzzle(this.photo, 3, 3);
661             }
```

```
662
663         }
664         //
665         // ================================================
666         // ----------------------------------------
667         // End playGame Handler
668         // ----------------------------------------
669         // ================================================
670     };
671
672     // ================================================
673     // Launch Slider Puzzle: (SWPA)
674     // - preferred game launch method for BOM.
675     // ================================================
676     document.addEventListener('DOMContentLoaded', function(){
677         game = new Phaser.Game(config); // standard Phaser III game launched
678         window.focus()
679         resize();
680         window.addEventListener("resize", resize, false);
681
682     }, false);
683
684
685     /* End of file */
686     /* Location: ./js/state/play.js */
687     ////////////////////////////////////////////////////////////////////////////
688     ////////////////////////////////////////////////////////////////////////////
689     // Original design. See Book review.
690     ////////////////////////////////////////////////////////////////////////////
691     ////////////////////////////////////////////////////////////////////////////
692     /**
693     // remove hard-coded variables
694     var PIECE_WIDTH = 200,  // image must be evenly divisible by piece size.
695         PIECE_HEIGHT = 200, // only square permitted
696         BOARD_COLS,         // = imageUsed / config.width
697         BOARD_ROWS;         // = imageUsed / config.height
698     var pzGrpGroup,     // puzzle grouped as Dynamic
699         pzGrpAmount,        // total pzGrp for victory
700         shuffledIndexArray = [];// recorded image shuffle
701     // abstract other variables below as required
702
703     function init(){
704         //  These are all set in the startPuzzle function
705         this.rows = 0;
706         this.columns = 0;
707
708         //  The width and height of each piece in the puzzle.
709         //  Again, this is set automatically in startPuzzle.
710         this.pieceWidth = 0;
711         this.pieceHeight = 0;
712
713         this.pzGrp = null;
714         this.spacer = null;
715
716         //  The speed at which the pzGrp slide, and the tween they use
717         this.slideSpeed = 250;
718         this.slideEase = 'Sine';
719
720         //  The number of iterations the puzzle walker will go through when
```



```
721         //  scrambling up the puzzle. 10 is a nice and easy puzzle, but
722         //  push it higher for much harder ones.
723         this.iterations = 10;
724 
725         //  The speed at which the pzGrp are shuffled at the start. This allows
726         //  the player to see the puzzle before trying to solve it. However if
727         //  you don't want this, just set the speed to zero and it'll appear
728         //  instantly 'scrambled'.
729         this.shuffleSpeed = 80;
730         this.shuffleEase = 'Linear';
731 
732         this.lastMove = null;
733 
734         //  The image in the Cache to be used for the puzzle.
735         //  Set in the startPuzzle function.
736         this.photo = '';
737 
738         this.action = GAMEAPP.ALLOW_CLICK;
739     }
740 
741     function preload() {
742         this.load.spritesheet("background",
            "assets/games/sliding-puzzle/bl.jpg", PIECE_WIDTH, PIECE_HEIGHT);
743 
744     }
745 
746     function create() {
747         // consolidated creation
748         // mixture of game data and visual display
749         // split into machanics vs. mechanisms
750         prepareBoard();
751     }
752 
753     function prepareBoard() {
754         // split into machanics vs. mechanisms
755         // working variables for build process
756         var pzGrpIndex = 0,
757             i, j,
758             piece;
759         // ==========
760         // move to game mechanics component
761         BOARD_COLS = Math.floor(config.width / PIECE_WIDTH);
762         BOARD_ROWS = Math.floor(config.height / PIECE_HEIGHT);
763         // game data stored in Game Mechanics
764         pzGrpAmount = BOARD_COLS * BOARD_ROWS;
765         // game data stored in Game Mechanics
766         shuffledIndexArray = createShuffledIndexArray();
767         // ==========
768         // game mechanism unique to JS Gaming Framework
769         // Phaser v2.x.x
770         pzGrpGroup = this.physics.add.staticGroup();
771         // Phaser III
772         // pzGrpGroup = this.physics.add.group();
773         // ======================
774         // Design Considerations:
775         // What should we Use in Phaser III??
776         //  - Static/Dynamic Groups, OR
777         //  - Static/Dynamic TileMaps OR
778         //  - "Containers" (an array of game objects) OR
```

```
779         //  - Grid?!
780         // ======================
781         // A Phaser III Group is a collection created to hold a bunch of the
            objects you want to control together. There two types of Group
            objects: Static or Dynamic. Static Groups don’t move (ground, walls,
            various obstacles), while Dynamic Groups have an active job in the
            game (avatars, bullets, enemies, etc). Once "children" objects are
            assigned into any group, you must call "refresh()" to update the
            group's bounding box (otherwise, the collisions will be checked
            against the default location, which is the top left corner of the
            scene).
782         // - Groups themselves aren't displayable, and can't be positioned,
            rotated, scaled, or hidden.
783         // -
            https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Container
            .html
784         // - Containers are used when you want two or more game objects to
            “behave as one flock”.
785         // - Container has position, angle, alpha, visible, but group does not.
786         // - Container controls properties of children, but group won't.
787         // -
            http://labs.phaser.io/edit.html?src=src\game%20objects\container\add%20
            array%20of%20sprites%20to%20container.js
788         // scene.physics.add.group(children, config);
789         // - A game object could be added to many group, but only added to a
            container (exclusive mode).
790         //
            https://rexrainbow.github.io/phaser3-rex-notes/docs/site/arcade-gameobj
            ect/#group
791         // https://rexrainbow.github.io/phaser3-rex-notes/docs/site/container/
792         //
            https://rexrainbow.github.io/phaser3-rex-notes/docs/site/containerlite/
            #children-group
793         // ======================
794         // A Phaser III Tilemap isn't a display object. It holds data about
            the map and can contain one or more layers, which are the display
            objects that actually render Tile objects. They come in two flavors:
            `StaticTilemapLayer` & `DynamicTilemapLayer`.
795         // - A `StaticTilemapLayer` is fast, but the tiles in that layer
            can’t be modified and can’t render per-tile effects like flipping or
            tint.
796         // * A `DynamicTilemapLayer` trades some speed for the flexibility
            and power of manipulating individual tiles. Unlike a Static Tilemap
            Layer, you can apply per-tile effects like tint or alpha, and you can
            change the tiles in a DynamicTilemapLayer.
797         // new DynamicTilemapLayer(scene, tilemap, layerIndex, tileset [, x]
            [, y])
798         //
799         //You can mix static and dynamic layers together in the same map.
800         // ======================
801         // Grid:
            https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Grid.html
802         // The Grid Shape is a Game Object that can be added to a Scene,
            Group or Container. You can treat it like any other Game Object in
            your game, such as tweening it, scaling it, or enabling it for input
            or physics. It provides a quick and easy way for you to render this
            shape in your game without using a texture, while still taking
            advantage of being fully batched in WebGL.
803         // This shape supports only fill colors and cannot be stroked.
```



```
804         // A Grid Shape allows you to display a grid in your game, where you
            can control the size of the grid as well as the width and height of
            the grid cells. You can set a fill color for each grid cell as well
            as an alternate fill color. When the alternate fill color is set then
            the grid cells will alternate the fill colors as they render,
            creating a chess-board effect. You can also optionally have an
            outline fill color. If set, this draws lines between the grid cells
            in the given color. If you specify an outline color with an alpha of
            zero, then it will draw the cells spaced out, but without the lines
            between them.
805         // ======================
806         // move to Game Mechanics Component
807         for (i = 0; i < BOARD_ROWS; i++){
808             for (j = 0; j < BOARD_COLS; j++){
809                 if (shuffledIndexArray[pzGrpIndex]) {
810                     // Phaser v2.x.x  display
811                     piece = pzGrpGroup.create(j * PIECE_WIDTH, i *
                        PIECE_HEIGHT, "background",
                        shuffledIndexArray[pzGrpIndex]);
812                     // into Phaser III Dynamic Group
813
814                 } else { //initial position of black piece
815                     // Phaser v2.x.x  display
816                     piece = pzGrpGroup.create(j * PIECE_WIDTH, i *
                        PIECE_HEIGHT);
817                     piece.black = true;
818                 }
819                 // about the Game Object (data):
820                 piece.name = 'piece' + i.toString() + 'x' + j.toString();
821                 // current grid location
822                 piece.currentIndex = pzGrpIndex;
823                 // final resting location
824                 piece.destIndex = shuffledIndexArray[pzGrpIndex];
825                 // Phaser v2.x.x
826                 piece.inputEnabled = true;
827                 // into Phaser III Dynamic Group
828                 // piece.setInteractive();
829                 // Game Object input responce from mouse/touch
830                 // Phaser v2.x.x
831                 piece.events.onInputDown.add(selectPiece, this);
832                 // into Phaser III
833                 // this.input.on("pointerdown", selectPiece, this);
834                 piece.posX = j;
835                 piece.posY = i;
836                 pzGrpIndex++;
837             }
838         }
839
840     }
841     // Phaser v2.x.x display mechanism
842     function selectPiece(piece) {
843
844         var blackPiece = canMove(piece);
845
846         //if there is a black piece in neighborhood
847         if (blackPiece) {
848             // Phaser v2.x.x  display
849             movePiece(piece, blackPiece);
850         }
```

```
851
852     }
853     // Phaser v2.x.x display mechanism
854     function canMove(piece) {
855
856         var foundBlackElem = false;
857         // Phaser v2.x.x display mechanism
858         pzGrpGroup.children.forEach(function(element) {
859             if (element.posX === (piece.posX - 1) && element.posY ===
                piece.posY && element.black ||
860                 element.posX === (piece.posX + 1) && element.posY ===
                    piece.posY && element.black ||
861                 element.posY === (piece.posY - 1) && element.posX ===
                    piece.posX && element.black ||
862                 element.posY === (piece.posY + 1) && element.posX ===
                    piece.posX && element.black) {
863                 foundBlackElem = element;
864                 return;
865             }
866         });
867
868         return foundBlackElem;
869     }
870     // Phaser v2.x.x display mechanism
871     function movePiece(piece, blackPiece) {
872
873         var tmpPiece = {
874             posX: piece.posX,
875             posY: piece.posY,
876             currentIndex: piece.currentIndex
877         };
878         // Phaser v2.x.x display mechanism - tween
879         game.add.tween(piece).to({x: blackPiece.posX * PIECE_WIDTH, y:
            blackPiece.posY * PIECE_HEIGHT}, 300, Phaser.Easing.Linear.None, true);
880
881         //change places of piece and blackPiece
882         piece.posX = blackPiece.posX;
883         piece.posY = blackPiece.posY;
884         piece.currentIndex = blackPiece.currentIndex;
885         piece.name ='piece' + piece.posX.toString() + 'x' +
            piece.posY.toString();
886
887         //piece is the new black
888         blackPiece.posX = tmpPiece.posX;
889         blackPiece.posY = tmpPiece.posY;
890         blackPiece.currentIndex = tmpPiece.currentIndex;
891         blackPiece.name ='piece' + blackPiece.posX.toString() + 'x' +
            blackPiece.posY.toString();
892
893         //after every move check if puzzle is completed
894
895         checkIfFinished();  // move to Game Mechanics Component
896     }
897
898     // move to Game Mechanics Component
899     function checkIfFinished() {
900
901         var isFinished = true;
902         // validates data from Game Mechanism
```



```
903         pzGrpGroup.children.forEach(function(element) {
904             if (element.currentIndex !== element.destIndex) {
905                 isFinished = false;
906                 return;
907             }
908         });
909 
910         if (isFinished) {
911             // tell Game Mechanism
912             showFinishedText();
913         }
914 
915     }
916     // Game Mechanism - display WON! or move to new "GameOver" Phase
917     function showFinishedText() {
918         // already in main.js
919         var style = { font: "40px Arial", fill: "#000", align: "center"};
920         // Phaser v2.x.x  display
921         var text = game.add.text(game.world.centerX, game.world.centerY,
            "Congratulations! \nYou made it!", style);
922         // into Phaser III
923         // var text = this.add.text(config.width/2, config.height/2,
            "Congratulations! \nYou made it!", style);
924         // Phaser v2.x.x  display
925         text.anchor.set(0.5);
926         // into Phaser III
927         // text.setOrigin(0.5);
928 
929     }
930     // move to Game Mechanics (data)
931     function createShuffledIndexArray() {
932 
933         var indexArray = [];
934 
935         for (var i = 0; i < pzGrpAmount; i++) {
936             indexArray.push(i);
937         }
938 
939         return shuffle(indexArray);
940 
941     }
942     // move to Game Mechanics (data)
943     function shuffle(array) {
944 
945         var counter = array.length,
946             temp,
947             index;
948 
949         while (counter > 0){
950             index = Math.floor(Math.random() * counter);
951 
952             counter--;
953             // standard Fisher Yates shuffle
954             // https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle
955             temp = array[counter];
956             array[counter] = array[index];
957             array[index] = temp;
958         }
959 
```

```
960         return array;
961
962     }
963     */
964
965     //
966     // ================================================
967     /* End of file */
968     /* Location: ./play.js */
969
```

